

# Package ‘Hmisc’

January 20, 2025

**Version** 5.2-2

**Date** 2025-01-10

**Title** Harrell Miscellaneous

**Maintainer** Frank E Harrell Jr <fh@fharrell.com>

**Depends** R (>= 4.2.0)

**Imports** methods, ggplot2, cluster, rpart, nnet, foreign, gtable, grid, gridExtra, data.table, htmlTable (>= 1.11.0), viridis, htmltools, base64enc, colorspace, rmarkdown, knitr, Formula

**Suggests** survival, qreport, acepack, chron, rms, mice, rstudioapi, tables, plotly (>= 4.5.6), rlang, plyr, VGAM, leaps, pcaPP, digest, parallel, polyspline, abind, kableExtra, rio, lattice, latticeExtra, gt, sparkline, jsonlite, htmlwidgets, qs, getPass, keyring, safer, htm2txt

**Description** Contains many functions useful for data analysis, high-level graphics, utility operations, functions for computing sample size and power, simulation, importing and annotating datasets, imputing missing values, advanced table making, variable clustering, character string manipulation, conversion of R objects to LaTeX and html code, recoding variables, caching, simplified parallel computing, encrypting and decrypting data using a safe workflow, general moving window statistical estimation, and assistance in interpreting principal component analysis.

**License** GPL (>= 2)

**LazyLoad** Yes

**URL** <https://hbiostat.org/R/Hmisc/>

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**NeedsCompilation** yes

**Author** Frank E Harrell Jr [aut, cre] (<<https://orcid.org/0000-0002-8271-5493>>), Charles Dupont [ctb] (contributed several functions and maintains latex functions)

**Repository** CRAN

**Date/Publication** 2025-01-10 23:30:03 UTC

## Contents

abs.error.pred . . . . .	6
addggLayers . . . . .	7
addMarginal . . . . .	9
all.is.numeric . . . . .	10
approxExtrap . . . . .	11
areg . . . . .	12
aregImpute . . . . .	16
binconf . . . . .	26
biVar . . . . .	27
bootkm . . . . .	30
bpower . . . . .	32
bpplot . . . . .	34
bystats . . . . .	36
capitalize . . . . .	38
ciapower . . . . .	39
cnvrt.coords . . . . .	40
colorFacet . . . . .	43
combine.levels . . . . .	44
combplotp . . . . .	45
completer . . . . .	47
consolidate . . . . .	49
contents . . . . .	50
cpower . . . . .	52
Cs . . . . .	55
csv.get . . . . .	56
curveRep . . . . .	58
cut2 . . . . .	63
data.frame.create.modify.check . . . . .	65
dataRep . . . . .	74
deff . . . . .	77
describe . . . . .	78
discrete . . . . .	85
dotchart2 . . . . .	87
dotchart3 . . . . .	89
dotchartpl . . . . .	93
ebpcomp . . . . .	97
Ecdf . . . . .	98
ecdfSteps . . . . .	102
equalBins . . . . .	103
errbar . . . . .	104
escapeRegex . . . . .	106
estSeqMarkovOrd . . . . .	107
estSeqSim . . . . .	110
event.chart . . . . .	112
event.convert . . . . .	123
event.history . . . . .	124

extractlabs . . . . .	129
flmport . . . . .	130
find.matches . . . . .	132
first.word . . . . .	136
format.df . . . . .	137
format.pval . . . . .	140
gbayes . . . . .	141
gbayesSeqSim . . . . .	148
getabd . . . . .	150
getHdata . . . . .	151
getRs . . . . .	152
getZip . . . . .	154
ggfreqScatter . . . . .	155
ggplotlyr . . . . .	157
GiniMd . . . . .	158
hashCheck . . . . .	159
hdquantile . . . . .	160
hidingTOC . . . . .	161
hist.data.frame . . . . .	163
histbackback . . . . .	164
histboxp . . . . .	165
hlab . . . . .	167
hlab . . . . .	167
hlab . . . . .	168
HmiscOverview . . . . .	169
hoeffd . . . . .	175
html . . . . .	177
htmltabv . . . . .	180
impute . . . . .	181
intMarkovOrd . . . . .	182
knitrSet . . . . .	184
labcurve . . . . .	186
label . . . . .	196
Lag . . . . .	201
latestFile . . . . .	202
latex . . . . .	203
latexCheckOptions . . . . .	212
latexDotchart . . . . .	213
latexTabular . . . . .	215
latexTherm . . . . .	216
legendfunctions . . . . .	218
list.tree . . . . .	218
makeNstr . . . . .	220
mApply . . . . .	220
mChoice . . . . .	222
mdb.get . . . . .	226
meltData . . . . .	227
Merge . . . . .	229
mgp.axis . . . . .	230

mhgr . . . . .	231
minor.tick . . . . .	233
Misc . . . . .	235
movStats . . . . .	240
mtitle . . . . .	243
multLines . . . . .	245
na.delete . . . . .	246
na.detail.response . . . . .	247
na.keep . . . . .	248
nCoincident . . . . .	249
nobsY . . . . .	250
nstr . . . . .	251
num.intercepts . . . . .	252
ordGroupBoot . . . . .	253
pairUpDiff . . . . .	254
panel.bpplot . . . . .	256
partition . . . . .	261
pc1 . . . . .	262
plot.princmp . . . . .	263
plotCorrM . . . . .	264
plotCorrPrecision . . . . .	265
plotlyM . . . . .	266
plsmo . . . . .	269
pMedian . . . . .	273
popower . . . . .	274
princmp . . . . .	278
print.char.list . . . . .	279
print.char.matrix . . . . .	281
print.princmp . . . . .	282
printL . . . . .	283
prnz . . . . .	284
prselect . . . . .	285
pstamp . . . . .	286
qcrypt . . . . .	287
qrxcenter . . . . .	288
r2describe . . . . .	289
R2Measures . . . . .	290
rcorr . . . . .	292
rcorr.cens . . . . .	294
rcorrp.cens . . . . .	296
rcspline.eval . . . . .	299
rcspline.plot . . . . .	301
rcspline.restate . . . . .	302
redun . . . . .	305
reShape . . . . .	307
rlegend . . . . .	310
rm.boot . . . . .	311
rMultinom . . . . .	320

runifChanged . . . . .	320
runParallel . . . . .	322
samplesize.bin . . . . .	323
sas.get . . . . .	324
sasxport.get . . . . .	331
Save . . . . .	334
scat1d . . . . .	335
score.binary . . . . .	343
sedit . . . . .	345
seqFreq . . . . .	348
show.pch . . . . .	348
showPsfrag . . . . .	349
simMarkovOrd . . . . .	350
simplifyDims . . . . .	352
simRegOrd . . . . .	353
smean.sd . . . . .	355
solvet . . . . .	357
somers2 . . . . .	357
soprobMarkovOrd . . . . .	359
soprobMarkovOrdM . . . . .	360
spikecomp . . . . .	361
spower . . . . .	363
spss.get . . . . .	369
src . . . . .	371
stata.get . . . . .	372
stat_plsmo . . . . .	373
string.bounding.box . . . . .	375
string.break.line . . . . .	375
stringDims . . . . .	376
subplot . . . . .	377
summarize . . . . .	379
summary.formula . . . . .	383
summaryM . . . . .	398
summaryP . . . . .	406
summaryRc . . . . .	411
summaryS . . . . .	413
symbol.freq . . . . .	419
sys . . . . .	420
t.test.cluster . . . . .	421
tabulr . . . . .	422
testCharDateTime . . . . .	425
tex . . . . .	426
transace . . . . .	427
transcan . . . . .	436
translate . . . . .	453
trunc.POSIXt . . . . .	454
units . . . . .	455
upData . . . . .	456

upFirst . . . . .	460
valueTags . . . . .	461
varclus . . . . .	463
vlab . . . . .	467
wtd.stats . . . . .	468
xtfrm.labelled . . . . .	472
xy.group . . . . .	472
xYplot . . . . .	473
yearDays . . . . .	482
ynbind . . . . .	483
%nin% . . . . .	484

<b>Index</b>	<b>485</b>
--------------	------------

---

abs.error.pred	<i>Indexes of Absolute Prediction Error for Linear Models</i>
----------------	---

---

## Description

Computes the mean and median of various absolute errors related to ordinary multiple regression models. The mean and median absolute errors correspond to the mean square due to regression, error, and total. The absolute errors computed are derived from  $\hat{Y} - \text{median}(\hat{Y})$ ,  $\hat{Y} - Y$ , and  $Y - \text{median}(Y)$ . The function also computes ratios that correspond to  $R^2$  and  $1 - R^2$  (but these ratios do not add to 1.0); the  $R^2$  measure is the ratio of mean or median absolute  $\hat{Y} - \text{median}(\hat{Y})$  to the mean or median absolute  $Y - \text{median}(Y)$ . The  $1 - R^2$  or SSE/SST measure is the mean or median absolute  $\hat{Y} - Y$  divided by the mean or median absolute  $\hat{Y} - \text{median}(\hat{Y})$ .

## Usage

```
abs.error.pred(fit, lp=NULL, y=NULL)

## S3 method for class 'abs.error.pred'
print(x, ...)
```

## Arguments

fit	a fit object typically from <code>lm</code> or <code>ols</code> that contains a y vector (i.e., you should have specified <code>y=TRUE</code> to the fitting function) unless the y argument is given to <code>abs.error.pred</code> . If you do not specify the lp argument, fit must contain <code>fitted.values</code> or <code>linear.predictors</code> . You must specify fit or both of lp and y.
lp	a vector of predicted values ( $\hat{Y}$ hat above) if fit is not given
y	a vector of response variable values if fit (with <code>y=TRUE</code> in effect) is not given
x	an object created by <code>abs.error.pred</code>
...	unused

**Value**

a list of class `abs.error.pred` (used by `print.abs.error.pred`) containing two matrices: differences and ratios.

**Author(s)**

Frank Harrell  
Department of Biostatistics  
Vanderbilt University School of Medicine  
<fh@fharrell.com>

**References**

Schemper M (2003): Stat in Med 22:2299-2308.  
Tian L, Cai T, Goetghebeur E, Wei LJ (2007): Biometrika 94:297-311.

**See Also**

[lm](#), [ols](#), [cor](#), [validate.ols](#)

**Examples**

```
set.seed(1)          # so can regenerate results
x1 <- rnorm(100)
x2 <- rnorm(100)
y  <- exp(x1+x2+rnorm(100))
f  <- lm(log(y) ~ x1 + poly(x2,3), y=TRUE)
abs.error.pred(lp=exp(fitted(f)), y=y)
rm(x1,x2,y,f)
```

---

addggLayers

*addggLayers*

---

**Description**

Add Spike Histograms and Extended Box Plots to ggplot

**Usage**

```
addggLayers(  
  g,  
  data,  
  type = c("ebp", "spike"),  
  ylim = layer_scales(g)$y$get_limits(),  
  by = "variable",  
  value = "value",  
  frac = 0.065,
```

```
    mult = 1,  
    facet = NULL,  
    pos = c("bottom", "top"),  
    showN = TRUE  
  )
```

### Arguments

<code>g</code>	a ggplot object
<code>data</code>	data frame/table containing raw data
<code>type</code>	specifies either extended box plot or spike histogram. Both are horizontal so are showing the distribution of the x-axis variable.
<code>ylim</code>	y-axis limits to use for scaling the height of the added plots, if you don't want to use the limits that ggplot has stored
<code>by</code>	the name of a variable in data used to stratify raw data
<code>value</code>	name of x-variable
<code>frac</code>	fraction of y-axis range to devote to vertical aspect of the added plot
<code>mult</code>	fudge factor for scaling y aspect
<code>facet</code>	optional faceting variable
<code>pos</code>	position for added plot
<code>showN</code>	sete to FALSE to not show sample sizes

### Details

For an example see [this](#). Note that it was not possible to just create the layers needed to be added, as creating these particular layers in isolation resulted in a ggplot error.

### Value

the original ggplot object with more layers added

### Author(s)

Frank Harrell

### See Also

`spikecomp()`



---

addMarginal	<i>Add Marginal Observations</i>
-------------	----------------------------------

---

### Description

Given a data frame and the names of variable, doubles the data frame for each variable with a new category "All" by default, or by the value of label. A new variable `.marginal.` is added to the resulting data frame, with value "" if the observation is an original one, and with value equal to the names of the variable being marginalized (separated by commas) otherwise. If there is another stratification variable besides the one in `...`, and that variable is nested inside the variable in `...`, specify `nested=variable name` to have the value of that variable set fo label whenever marginal observations are created for `....` See the state-city example below.

### Usage

```
addMarginal(data, ..., label = "All", margloc=c('last', 'first'), nested)
```

### Arguments

<code>data</code>	a data frame
<code>...</code>	a list of names of variables to marginalize
<code>label</code>	category name for added marginal observations
<code>margloc</code>	location for marginal category within factor variable specifying categories. Set to "first" to override the default - to put a category with value label as the first category.
<code>nested</code>	a single unquoted variable name if used

### Examples

```
d <- expand.grid(sex=c('female', 'male'), country=c('US', 'Romania'),
               reps=1:2)
addMarginal(d, sex, country)

# Example of nested variables
d <- data.frame(state=c('AL', 'AL', 'GA', 'GA', 'GA'),
               city=c('Mobile', 'Montgomery', 'Valdosto',
                     'Augusta', 'Atlanta'),
               x=1:5, stringsAsFactors=TRUE)
addMarginal(d, state, nested=city) # cite set to 'All' when state is
```

---

all.is.numeric

*Check if All Elements in Character Vector are Numeric*


---

### Description

Tests, without issuing warnings, whether all elements of a character vector are legal numeric values, or optionally converts the vector to a numeric vector. Leading and trailing blanks in x are ignored.

### Usage

```
all.is.numeric(x, what = c("test", "vector", "nonnum"), extras=c('.', 'NA'))
```

### Arguments

x	a character vector
what	specify what="vector" to return a numeric vector if it passes the test, or the original character vector otherwise, the default "test" to return FALSE if there are no non-missing non-extra values of x or there is at least one non-numeric value of x, or "nonnum" to return the vector of non-extra, non-NA, non-numeric values of x.
extras	a vector of character strings to count as numeric values, other than ".".

### Value

a logical value if what="test" or a vector otherwise

### Author(s)

Frank Harrell

### See Also

[as.numeric](#)

### Examples

```
all.is.numeric(c('1', '1.2', '3'))
all.is.numeric(c('1', '1.2', '3a'))
all.is.numeric(c('1', '1.2', '3'), 'vector')
all.is.numeric(c('1', '1.2', '3a'), 'vector')
all.is.numeric(c('1', '', ' '), 'vector')
all.is.numeric(c('1', '1.2', '3a'), 'nonnum')
```

---

`approxExtrap`*Linear Extrapolation*

---

**Description**

Works in conjunction with the [approx](#) function to do linear extrapolation. [approx](#) in R does not support extrapolation at all, and it is buggy in S-Plus 6.

**Usage**

```
approxExtrap(x, y, xout, method = "linear", n = 50, rule = 2, f = 0,
             ties = "ordered", na.rm = FALSE)
```

**Arguments**

`x`, `y`, `xout`, `method`, `n`, `rule`, `f`  
see [approx](#)

`ties` applies only to R. See [approx](#)

`na.rm` set to TRUE to remove NAs in `x` and `y` before proceeding

**Details**

Duplicates in `x` (and corresponding `y` elements) are removed before using [approx](#).

**Value**

a vector the same length as `xout`

**Author(s)**

Frank Harrell

**See Also**

[approx](#)

**Examples**

```
approxExtrap(1:3, 1:3, xout=c(0,4))
```

areg

*Additive Regression with Optimal Transformations on Both Sides using Canonical Variates*

## Description

Expands continuous variables into restricted cubic spline bases and categorical variables into dummy variables and fits a multivariate equation using canonical variates. This finds optimum transformations that maximize  $R^2$ . Optionally, the bootstrap is used to estimate the covariance matrix of both left- and right-hand-side transformation parameters, and to estimate the bias in the  $R^2$  due to overfitting and compute the bootstrap optimism-corrected  $R^2$ . Cross-validation can also be used to get an unbiased estimate of  $R^2$  but this is not as precise as the bootstrap estimate. The bootstrap and cross-validation may also be used to get estimates of mean and median absolute error in predicted values on the original y scale. These two estimates are perhaps the best ones for gauging the accuracy of a flexible model, because it is difficult to compare  $R^2$  under different y-transformations, and because  $R^2$  allows for an out-of-sample recalibration (i.e., it only measures relative errors).

Note that uncertainty about the proper transformation of y causes an enormous amount of model uncertainty. When the transformation for y is estimated from the data a high variance in predicted values on the original y scale may result, especially if the true transformation is linear. Comparing bootstrap or cross-validated mean absolute errors with and without restricted the y transform to be linear (ytype='l') may help the analyst choose the proper model complexity.

## Usage

```
areg(x, y, xtype = NULL, ytype = NULL, nk = 4,
     B = 0, na.rm = TRUE, tolerance = NULL, crossval = NULL)
```

```
## S3 method for class 'areg'
print(x, digits=4, ...)
```

```
## S3 method for class 'areg'
plot(x, whichx = 1:ncol(x$x), ...)
```

```
## S3 method for class 'areg'
predict(object, x, type=c('lp','fitted','x'),
        what=c('all','sample'), ...)
```

## Arguments

x	A single predictor or a matrix of predictors. Categorical predictors are required to be coded as integers (as factor does internally). For predict, x is a data matrix with the same integer codes that were originally used for categorical variables.
y	a factor, categorical, character, or numeric response variable
xtype	a vector of one-letter character codes specifying how each predictor is to be modeled, in order of columns of x. The codes are "s" for smooth function

	(using restricted cubic splines), "1" for no transformation (linear), or "c" for categorical (to cause expansion into dummy variables). Default is "s" if $nk > 0$ and "1" if $nk=0$ .
ytype	same coding as for xtype. Default is "s" for a numeric variable with more than two unique values, "1" for a binary numeric variable, and "c" for a factor, categorical, or character variable.
nk	number of knots, 0 for linear, or 3 or more. Default is 4 which will fit 3 parameters to continuous variables (one linear term and two nonlinear terms)
B	number of bootstrap resamples used to estimate covariance matrices of transformation parameters. Default is no bootstrapping.
na.rm	set to FALSE if you are sure that observations with NAs have already been removed
tolerance	singularity tolerance. List source code for <code>lm.fit.qr.bare</code> for details.
crossval	set to a positive integer k to compute k-fold cross-validated R-squared (square of first canonical correlation) and mean and median absolute error of predictions on the original scale
digits	number of digits to use in formatting for printing
object	an object created by areg
whichx	integer or character vector specifying which predictors are to have their transformations plotted (default is all). The y transformation is always plotted.
type	tells predict whether to obtain predicted untransformed y (type='lp', the default) or predicted y on the original scale (type='fitted'), or the design matrix for the right-hand side (type='x').
what	When the y-transform is non-monotonic you may specify what='sample' to predict to obtain a random sample of y values on the original scale instead of a matrix of all y-inverses. See <a href="#">inverseFunction</a> .
...	arguments passed to the plot function.

## Details

areg is a competitor of ace in the acepack package. Transformations from ace are seldom smooth enough and are often overfitted. With areg the complexity can be controlled with the nk parameter, and predicted values are easy to obtain because parametric functions are fitted.

If one side of the equation has a categorical variable with more than two categories and the other side has a continuous variable not assumed to act linearly, larger sample sizes are needed to reliably estimate transformations, as it is difficult to optimally score categorical variables to maximize  $R^2$  against a simultaneously optimally transformed continuous variable.

## Value

a list of class "areg" containing many objects

**Author(s)**

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>

**References**

Breiman and Friedman, Journal of the American Statistical Association (September, 1985).

**See Also**

[cancor](#), [ace](#), [transcan](#)

**Examples**

```
set.seed(1)

ns <- c(30,300,3000)
for(n in ns) {
  y <- sample(1:5, n, TRUE)
  x <- abs(y-3) + runif(n)
  par(mfrow=c(3,4))
  for(k in c(0,3:5)) {
    z <- areg(x, y, ytype='c', nk=k)
    plot(x, z$tx)
    title(paste('R2=', format(z$rsquared)))
    tapply(z$ty, y, range)
    a <- tapply(x,y,mean)
    b <- tapply(z$ty,y,mean)
    plot(a,b)
    abline(lsfite(a,b))
    # Should get same result to within linear transformation if reverse x and y
    w <- areg(y, x, xtype='c', nk=k)
    plot(z$ty, w$tx)
    title(paste('R2=', format(w$rsquared)))
    abline(lsfite(z$ty, w$tx))
  }
}

par(mfrow=c(2,2))
# Example where one category in y differs from others but only in variance of x
n <- 50
y <- sample(1:5,n,TRUE)
x <- rnorm(n)
x[y==1] <- rnorm(sum(y==1), 0, 5)
z <- areg(x,y,xtype='l',ytype='c')
z
plot(z)
z <- areg(x,y,ytype='c')
z
plot(z)
```

```

## Not run:
# Examine overfitting when true transformations are linear
par(mfrow=c(4,3))
for(n in c(200,2000)) {
  x <- rnorm(n); y <- rnorm(n) + x
  for(nk in c(0,3,5)) {
    z <- areg(x, y, nk=nk, crossval=10, B=100)
    print(z)
    plot(z)
    title(paste('n=',n))
  }
}
par(mfrow=c(1,1))

# Underfitting when true transformation is quadratic but overfitting
# when y is allowed to be transformed
set.seed(49)
n <- 200
x <- rnorm(n); y <- rnorm(n) + .5*x^2
#areg(x, y, nk=0, crossval=10, B=100)
#areg(x, y, nk=4, ytype='l', crossval=10, B=100)
z <- areg(x, y, nk=4) #, crossval=10, B=100)
z
# Plot x vs. predicted value on original scale. Since y-transform is
# not monotonic, there are multiple y-inverses
xx <- seq(-3.5,3.5,length=1000)
yhat <- predict(z, xx, type='fitted')
plot(x, y, xlim=c(-3.5,3.5))
for(j in 1:ncol(yhat)) lines(xx, yhat[,j], col=j)
# Plot a random sample of possible y inverses
yhats <- predict(z, xx, type='fitted', what='sample')
points(xx, yhats, pch=2)

## End(Not run)

# True transformation of x1 is quadratic, y is linear
n <- 200
x1 <- rnorm(n); x2 <- rnorm(n); y <- rnorm(n) + x1^2
z <- areg(cbind(x1,x2),y,xtype=c('s','l'),nk=3)
par(mfrow=c(2,2))
plot(z)

# y transformation is inverse quadratic but areg gets the same answer by
# making x1 quadratic
n <- 5000
x1 <- rnorm(n); x2 <- rnorm(n); y <- (x1 + rnorm(n))^2
z <- areg(cbind(x1,x2),y,nk=5)
par(mfrow=c(2,2))
plot(z)

# Overfit 20 predictors when no true relationships exist
n <- 1000

```

```

x <- matrix(runif(n*20),n,20)
y <- rnorm(n)
z <- areg(x, y, nk=5) # add crossval=4 to expose the problem

# Test predict function
n <- 50
x <- rnorm(n)
y <- rnorm(n) + x
g <- sample(1:3, n, TRUE)
z <- areg(cbind(x,g),y,xtype=c('s','c'))
range(predict(z, cbind(x,g)) - z$linear.predictors)

```

---

aregImpute

---

*Multiple Imputation using Additive Regression, Bootstrapping, and Predictive Mean Matching*


---

## Description

The `transcan` function creates flexible additive imputation models but provides only an approximation to true multiple imputation as the imputation models are fixed before all multiple imputations are drawn. This ignores variability caused by having to fit the imputation models. `aregImpute` takes all aspects of uncertainty in the imputations into account by using the bootstrap to approximate the process of drawing predicted values from a full Bayesian predictive distribution. Different bootstrap resamples are used for each of the multiple imputations, i.e., for the  $i$ th imputation of a sometimes missing variable,  $i=1, 2, \dots, n$ . `impute`, a flexible additive model is fitted on a sample with replacement from the original data and this model is used to predict all of the original missing and non-missing values for the target variable.

`areg` is used to fit the imputation models. By default, linearity is assumed for target variables (variables being imputed) and  $nk=3$  knots are assumed for continuous predictors transformed using restricted cubic splines. If  $nk$  is three or greater and `tlinear` is set to `FALSE`, `areg` simultaneously finds transformations of the target variable and of all of the predictors, to get a good fit assuming additivity, maximizing  $R^2$ , using the same canonical correlation method as `transcan`. Flexible transformations may be overridden for specific variables by specifying the identity transformation for them. When a categorical variable is being predicted, the flexible transformation is Fisher's optimum scoring method. Nonlinear transformations for continuous variables may be nonmonotonic. If  $nk$  is a vector, `areg`'s bootstrap and `crossval=10` options will be used to help find the optimum validating value of  $nk$  over values of that vector, at the last imputation iteration. For the imputations, the minimum value of  $nk$  is used.

Instead of defaulting to taking random draws from fitted imputation models using random residuals as is done by `transcan`, `aregImpute` by default uses predictive mean matching with optional weighted probability sampling of donors rather than using only the closest match. Predictive mean matching works for binary, categorical, and continuous variables without the need for iterative maximum likelihood fitting for binary and categorical variables, and without the need for computing residuals or for curtailing imputed values to be in the range of actual data. Predictive mean matching is especially attractive when the variable being imputed is also being transformed automatically. Constraints may be placed on variables being imputed with predictive mean matching, e.g., a missing hospital discharge date may be required to be imputed from a donor observation



whose discharge date is before the recipient subject's first post-discharge visit date. See Details below for more information about the algorithm. A "regression" method is also available that is similar to that used in `transcan`. This option should be used when mechanistic missingness requires the use of extrapolation during imputation.

A print method summarizes the results, and a plot method plots distributions of imputed values. Typically, `fit.mult.impute` will be called after `aregImpute`.

If a target variable is transformed nonlinearly (i.e., if `nk` is greater than zero and `tlinear` is set to `FALSE`) and the estimated target variable transformation is non-monotonic, imputed values are not unique. When `type='regression'`, a random choice of possible inverse values is made.

The `reformM` function provides two ways of recreating a formula to give to `aregImpute` by reordering the variables in the formula. This is a modified version of a function written by Yong Hao Pua. One can specify `nperm` to obtain a list of `nperm` randomly permuted variables. The list is converted to a single ordinary formula if `nperm=1`. If `nperm` is omitted, variables are sorted in descending order of the number of NAs. `reformM` also prints a recommended number of multiple imputations to use, which is a minimum of 5 and the percent of incomplete observations.

## Usage

```
aregImpute(formula, data, subset, n.impute=5, group=NULL,
           nk=3, tlinear=TRUE, type=c('pmm','regression','normpmm'),
           pmmtyp=1, match=c('weighted','closest','kclosest'),
           kclosest=3, fweighted=0.2,
           curtail=TRUE, constraint=NULL,
           boot.method=c('simple','approximate bayesian'),
           burnin=3, x=FALSE, pr=TRUE, plotTrans=FALSE, tolerance=NULL, B=75)
## S3 method for class 'aregImpute'
print(x, digits=3, ...)
## S3 method for class 'aregImpute'
plot(x, nclass=NULL, type=c('ecdf','hist'),
     datadensity=c("hist", "none", "rug", "density"),
     diagnostics=FALSE, maxn=10, ...)
reformM(formula, data, nperm)
```

## Arguments

- |                      |  |
|----------------------|--|
| <code>formula</code> | an S model formula. You can specify restrictions for transformations of variables. The function automatically determines which variables are categorical (i.e., factor, category, or character vectors). Binary variables are automatically restricted to be linear. Force linear transformations of continuous variables by enclosing variables by the <code>identify</code> function ( <code>I()</code> ). It is recommended that <code>factor()</code> or <code>as.factor()</code> do not appear in the formula but instead variables be converted to factors as needed and stored in the data frame. That way imputations for factor variables (done using <code>impute.transcan</code> for example) will be correct. Currently <code>reformM</code> does not handle variables that are enclosed in functions such as <code>I()</code> . |
| <code>x</code>       | an object created by <code>aregImpute</code> . For <code>aregImpute</code> , set <code>x</code> to <code>TRUE</code> to save the data matrix containing the final (number <code>n.impute</code> ) imputations in the result.   |

	This is needed if you want to later do out-of-sample imputation. Categorical variables are coded as integers in this matrix.
data	input raw data
subset	These may be also be specified. You may not specify <code>na.action</code> as <code>na.retain</code> is always used.
n.impute	number of multiple imputations. <code>n.impute=5</code> is frequently recommended but 10 or more doesn't hurt.
group	a character or factor variable the same length as the number of observations in data and containing no NAs. When <code>group</code> is present, causes a bootstrap sample of the observations corresponding to non-NAs of a target variable to have the same frequency distribution of <code>group</code> as the that in the non-NAs of the original sample. This can handle k-sample problems as well as lower the chance that a bootstrap sample will have a missing cell when the original cell frequency was low.
nk	number of knots to use for continuous variables. When both the target variable and the predictors are having optimum transformations estimated, there is more instability than with normal regression so the complexity of the model should decrease more sharply as the sample size decreases. Hence set <code>nk</code> to 0 (to force linearity for non-categorical variables) or 3 (minimum number of knots possible with a linear tail-restricted cubic spline) for small sample sizes. Simulated problems as in the examples section can assist in choosing <code>nk</code> . Set <code>nk</code> to a vector to get bootstrap-validated and 10-fold cross-validated $R^2$ and mean and median absolute prediction errors for imputing each sometimes-missing variable, with <code>nk</code> ranging over the given vector. The errors are on the original untransformed scale. The mean absolute error is the recommended basis for choosing the number of knots (or linearity).
tlinear	set to FALSE to allow a target variable (variable being imputed) to have a nonlinear left-hand-side transformation when <code>nk</code> is 3 or greater
type	The default is "pmm" for predictive mean matching, which is a more nonparametric approach that will work for categorical as well as continuous predictors. Alternatively, use "regression" when all variables that are sometimes missing are continuous and the missingness mechanism is such that entire intervals of population values are unobserved. See the Details section for more information. Another method, <code>type="normpmm"</code> , only works when variables containing NAs are continuous and <code>tlinear</code> is TRUE (the default), meaning that the variable being imputed is not transformed when it is on the left hand model side. <code>normpmm</code> assumes that the imputation regression parameter estimates are multivariately normally distributed and that the residual variance has a scaled chi-squared distribution. For each imputation a random draw of the estimates is taken and a random draw from <code>sigma</code> is combined with those to get a random draw from the posterior predicted value distribution. Predictive mean matching is then done matching these predicted values from incomplete observations with predicted values from complete potential donor observations, where the latter predictions are based on the imputation model least squares parameter estimates and not on random draws from the posterior. For the plot method, specify <code>type="hist"</code> to draw histograms of imputed values with rug plots at the top, or <code>type="ecdf"</code> (the default) to draw empirical CDFs with spike histograms at the bottom.

pmmttype	type of matching to be used for predictive mean matching when <code>type="pmm"</code> . <code>pmmttype=2</code> means that predicted values for both target incomplete and complete observations come from a fit from the same bootstrap sample. <code>pmmttype=1</code> , the default, means that predicted values for complete observations are based on additive regression fits on original complete observations (using last imputations for non-target variables as with the other methods), and using fits on a bootstrap sample to get predicted values for missing target variables. See van Buuren (2012) section 3.4.2 where <code>pmmttype=1</code> is said to work much better when the number of variables is small. <code>pmmttype=3</code> means that complete observation predicted values come from a bootstrap sample fit whereas target incomplete observation predicted values come from a sample with replacement from the bootstrap fit (approximate Bayesian bootstrap).
match	Defaults to <code>match="weighted"</code> to do weighted multinomial probability sampling using the tricube function (similar to <code>lowess</code> ) as the weights. The argument of the tricube function is the absolute difference in transformed predicted values of all the donors and of the target predicted value, divided by a scaling factor. The scaling factor in the tricube function is <code>fweighted</code> times the mean absolute difference between the target predicted value and all the possible donor predicted values. Set <code>match="closest"</code> to find as the donor the observation having the closest predicted transformed value, even if that same donor is found repeatedly. Set <code>match="kclosest"</code> to use a slower implementation that finds, after jittering the complete case predicted values, the <code>kclosest</code> complete cases on the target variable being imputed, then takes a random sample of one of these <code>kclosest</code> cases.
kclosest	see <code>match</code>
fweighted	Smoothing parameter (multiple of mean absolute difference) used when <code>match="weighted"</code> , with a default value of 0.2. Set <code>fweighted</code> to a number between 0.02 and 0.2 to force the donor to have a predicted value closer to the target, and set <code>fweighted</code> to larger values (but seldom larger than 1.0) to allow donor values to be less tightly matched. See the examples below to learn how to study the relationship between <code>fweighted</code> and the standard deviation of multiple imputations within individuals.
curtail	applies if <code>type='regression'</code> , causing imputed values to be curtailed at the observed range of the target variable. Set to <code>FALSE</code> to allow extrapolation outside the data range.
constraint	for predictive mean matching <code>constraint</code> is a named list specifying <code>R expression()</code> s encoding constraints on which donor observations are allowed to be used, based on variables that are not missing, i.e., based on donor observations and/or recipient observations as long as the target variable being imputed is not used for the recipients. The expressions must evaluate to a logical vector with no NAs and whose length is the number of rows in the donor observations. The expressions refer to donor observations by prefixing variable names by <code>d\$</code> , and to a single recipient observation by prefixing variables names by <code>r\$</code> .
boot.method	By default, simple bootstrapping is used in which the target variable is predicted using a sample with replacement from the observations with non-missing target variable. Specify <code>boot.method='approximate bayesian'</code> to build the imputation models from a sample with replacement from a sample with replacement of

	the observations with non-missing targets. Preliminary simulations have shown this results in good confidence coverage of the final model parameters when <code>type='regression'</code> is used. Not implemented when <code>group</code> is used.
<code>burnin</code>	<code>aregImpute</code> does <code>burnin + n.impute</code> iterations of the entire modeling process. The first <code>burnin</code> imputations are discarded. More burn-in iterations may be required when multiple variables are missing on the same observations. When only one variable is missing, no burn-ins are needed and <code>burnin</code> is set to zero if unspecified.
<code>pr</code>	set to <code>FALSE</code> to suppress printing of iteration messages
<code>plotTrans</code>	set to <code>TRUE</code> to plot <code>ace</code> or <code>avas</code> transformations for each variable for each of the multiple imputations. This is useful for determining whether transformations are reasonable. If transformations are too noisy or have long flat sections (resulting in "lumps" in the distribution of imputed values), it may be advisable to place restrictions on the transformations (monotonicity or linearity).
<code>tolerance</code>	singularity criterion; list the source code in the <code>lm.fit.qr.bare</code> function for details
<code>B</code>	number of bootstrap resamples to use if <code>nk</code> is a vector
<code>digits</code>	number of digits for printing
<code>nclass</code>	number of bins to use in drawing histogram
<code>datadensity</code>	see <a href="#">Ecdf</a>
<code>diagnostics</code>	Specify <code>diagnostics=TRUE</code> to draw plots of imputed values against sequential imputation numbers, separately for each missing observations and variable.
<code>maxn</code>	Maximum number of observations shown for diagnostics. Default is <code>maxn=10</code> , which limits the number of observations plotted to at most the first 10.
<code>nperm</code>	number of random formula permutations for <code>reformM</code> ; omit to sort variables by descending missing count.
<code>...</code>	other arguments that are ignored

## Details

The sequence of steps used by the `aregImpute` algorithm is the following.

- (1) For each variable containing  $m$  NAs where  $m > 0$ , initialize the NAs to values from a random sample (without replacement if a sufficient number of non-missing values exist) of size  $m$  from the non-missing values.
- (2) For `burnin+n.impute` iterations do the following steps. The first `burnin` iterations provide a burn-in, and imputations are saved only from the last `n.impute` iterations.
- (3) For each variable containing any NAs, draw a sample with replacement from the observations in the entire dataset in which the current variable being imputed is non-missing. Fit a flexible additive model to predict this target variable while finding the optimum transformation of it (unless the identity transformation is forced). Use this fitted flexible model to predict the target variable in all of the original observations. Impute each missing value of the target variable with the observed value whose predicted transformed value is closest to the predicted transformed value of the missing value (if `match="closest"` and `type="pmm"`), or use a draw from a multinomial distribution with probabilities derived from distance weights, if `match="weighted"` (the default).

(4) After these imputations are computed, use these random draw imputations the next time the current target variable is used as a predictor of other sometimes-missing variables.

When `match="closest"`, predictive mean matching does not work well when fewer than 3 variables are used to predict the target variable, because many of the multiple imputations for an observation will be identical. In the extreme case of one right-hand-side variable and assuming that only monotonic transformations of left and right-side variables are allowed, every bootstrap resample will give predicted values of the target variable that are monotonically related to predicted values from every other bootstrap resample. The same is true for Bayesian predicted values. This causes predictive mean matching to always match on the same donor observation.

When the missingness mechanism for a variable is so systematic that the distribution of observed values is truncated, predictive mean matching does not work. It will only yield imputed values that are near observed values, so intervals in which no values are observed will not be populated by imputed values. For this case, the only hope is to make regression assumptions and use extrapolation. With `type="regression"`, `aregImpute` will use linear extrapolation to obtain a (hopefully) reasonable distribution of imputed values. The `"regression"` option causes `aregImpute` to impute missing values by adding a random sample of residuals (with replacement if there are more NAs than measured values) on the transformed scale of the target variable. After random residuals are added, predicted random draws are obtained on the original untransformed scale using reverse linear interpolation on the table of original and transformed target values (linear extrapolation when a random residual is large enough to put the random draw prediction outside the range of observed values). The bootstrap is used as with `type="pmm"` to factor in the uncertainty of the imputation model.

As model uncertainty is high when the transformation of a target variable is unknown, `tlinear` defaults to `TRUE` to limit the variance in predicted values when `nk` is positive.

## Value

a list of class `"aregImpute"` containing the following elements:

<code>call</code>	the function call expression
<code>formula</code>	the formula specified to <code>aregImpute</code>
<code>match</code>	the match argument
<code>fweighted</code>	the <code>fweighted</code> argument
<code>n</code>	total number of observations in input dataset
<code>p</code>	number of variables
<code>na</code>	list of subscripts of observations for which values were originally missing
<code>nna</code>	named vector containing the numbers of missing values in the data
<code>type</code>	vector of types of transformations used for each variable (" <code>s</code> ", " <code>l</code> ", " <code>c</code> " for smooth spline, linear, or categorical with dummy variables)
<code>tlinear</code>	value of <code>tlinear</code> parameter
<code>nk</code>	number of knots used for smooth transformations
<code>cat.levels</code>	list containing character vectors specifying the levels of categorical variables
<code>df</code>	degrees of freedom (number of parameters estimated) for each variable
<code>n.impute</code>	number of multiple imputations per missing value

imputed	a list containing matrices of imputed values in the same format as those created by <code>transcan</code> . Categorical variables are coded using their integer codes. Variables having no missing values will have NULL matrices in the list.
x	if x is TRUE, the original data matrix with integer codes for categorical variables
rsq	for the last round of imputations, a vector containing the R-squares with which each sometimes-missing variable could be predicted from the others by <code>ace</code> or <code>avas</code> .

### Author(s)

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>

### References

- van Buuren, Stef. Flexible Imputation of Missing Data. Chapman & Hall/CRC, Boca Raton FL, 2012.
- Little R, An H. Robust likelihood-based analysis of multivariate data with missing values. *Statistica Sinica* 14:949-968, 2004.
- van Buuren S, Brand JPL, Groothuis-Oudshoorn CGM, Rubin DB. Fully conditional specifications in multivariate imputation. *J Stat Comp Sim* 72:1049-1064, 2006.
- de Groot JAH, Janssen KJM, Zwinderman AH, Moons KGM, Reitsma JB. Multiple imputation to correct for partial verification bias revisited. *Stat Med* 27:5880-5889, 2008.
- Siddique J. Multiple imputation using an iterative hot-deck with distance-based donor selection. *Stat Med* 27:83-102, 2008.
- White IR, Royston P, Wood AM. Multiple imputation using chained equations: Issues and guidance for practice. *Stat Med* 30:377-399, 2011.
- Curnow E, Carpenter JR, Heron JE, et al: Multiple imputation of missing data under missing at random: compatible imputation models are not sufficient to avoid bias if they are mis-specified. *J Clin Epi* June 9, 2023. DOI:10.1016/j.jclinepi.2023.06.011.

### See Also

[fit.mult.impute](#), [transcan](#), [areg](#), [naclus](#), [naplot](#), [mice](#), [dotchart3](#), [Ecdf](#), [completer](#)

### Examples

```
# Check that aregImpute can almost exactly estimate missing values when
# there is a perfect nonlinear relationship between two variables
# Fit restricted cubic splines with 4 knots for x1 and x2, linear for x3
set.seed(3)
x1 <- rnorm(200)
x2 <- x1^2
x3 <- runif(200)
m <- 30
x2[1:m] <- NA
```

```

a <- aregImpute(~x1+x2+I(x3), n.impute=5, nk=4, match='closest')
a
matplot(x1[1:m]^2, a$imputed$x2)
abline(a=0, b=1, lty=2)

x1[1:m]^2
a$imputed$x2

# Multiple imputation and estimation of variances and covariances of
# regression coefficient estimates accounting for imputation
# Example 1: large sample size, much missing data, no overlap in
# NAs across variables
x1 <- factor(sample(c('a','b','c'),1000,TRUE))
x2 <- (x1=='b') + 3*(x1=='c') + rnorm(1000,0,2)
x3 <- rnorm(1000)
y <- x2 + 1*(x1=='c') + .2*x3 + rnorm(1000,0,2)
orig.x1 <- x1[1:250]
orig.x2 <- x2[251:350]
x1[1:250] <- NA
x2[251:350] <- NA
d <- data.frame(x1,x2,x3,y, stringsAsFactors=TRUE)
# Find value of nk that yields best validating imputation models
# tlinear=FALSE means to not force the target variable to be linear
f <- aregImpute(~y + x1 + x2 + x3, nk=c(0,3:5), tlinear=FALSE,
               data=d, B=10) # normally B=75
f
# Try forcing target variable (x1, then x2) to be linear while allowing
# predictors to be nonlinear (could also say tlinear=TRUE)
f <- aregImpute(~y + x1 + x2 + x3, nk=c(0,3:5), data=d, B=10)
f

## Not run:
# Use 100 imputations to better check against individual true values
f <- aregImpute(~y + x1 + x2 + x3, n.impute=100, data=d)
f
par(mfrow=c(2,1))
plot(f)
modecat <- function(u) {
  tab <- table(u)
  as.numeric(names(tab)[tab==max(tab)][1])
}
table(orig.x1,apply(f$imputed$x1, 1, modecat))
par(mfrow=c(1,1))
plot(orig.x2, apply(f$imputed$x2, 1, mean))
fmi <- fit.mult.impute(y ~ x1 + x2 + x3, lm, f,
                     data=d)
sqrt(diag(vcov(fmi)))
fcc <- lm(y ~ x1 + x2 + x3)
summary(fcc) # SEs are larger than from mult. imputation

## End(Not run)
## Not run:

```

```

# Example 2: Very discriminating imputation models,
# x1 and x2 have some NAs on the same rows, smaller n
set.seed(5)
x1 <- factor(sample(c('a','b','c'),100,TRUE))
x2 <- (x1=='b') + 3*(x1=='c') + rnorm(100,0,.4)
x3 <- rnorm(100)
y <- x2 + 1*(x1=='c') + .2*x3 + rnorm(100,0,.4)
orig.x1 <- x1[1:20]
orig.x2 <- x2[18:23]
x1[1:20] <- NA
x2[18:23] <- NA
#x2[21:25] <- NA
d <- data.frame(x1,x2,x3,y, stringsAsFactors=TRUE)
n <- naclus(d)
plot(n); naplot(n) # Show patterns of NAs
# 100 imputations to study them; normally use 5 or 10
f <- aregImpute(~y + x1 + x2 + x3, n.impute=100, nk=0, data=d)
par(mfrow=c(2,3))
plot(f, diagnostics=TRUE, maxn=2)
# Note: diagnostics=TRUE makes graphs similar to those made by:
# r <- range(f$imputed$x2, orig.x2)
# for(i in 1:6) { # use 1:2 to mimic maxn=2
#   plot(1:100, f$imputed$x2[i,], ylim=r,
#     ylab=paste("Imputations for Obs.",i))
#   abline(h=orig.x2[i],lty=2)
# }

table(orig.x1,apply(f$imputed$x1, 1, modecat))
par(mfrow=c(1,1))
plot(orig.x2, apply(f$imputed$x2, 1, mean))

fmi <- fit.mult.impute(y ~ x1 + x2, lm, f,
  data=d)
sqrt(diag(vcov(fmi)))
fcc <- lm(y ~ x1 + x2)
summary(fcc) # SEs are larger than from mult. imputation

## End(Not run)

## Not run:
# Study relationship between smoothing parameter for weighting function
# (multiplier of mean absolute distance of transformed predicted
# values, used in tricube weighting function) and standard deviation
# of multiple imputations. SDs are computed from average variances
# across subjects. match="closest" same as match="weighted" with
# small value of fweightd.
# This example also shows problems with predicted mean
# matching almost always giving the same imputed values when there is
# only one predictor (regression coefficients change over multiple
# imputations but predicted values are virtually 1-1 functions of each
# other)

```



```

set.seed(23)
x <- runif(200)
y <- x + runif(200, -.05, .05)
r <- resid(lsfitt(x,y))
rmse <- sqrt(sum(r^2)/(200-2)) # sqrt of residual MSE

y[1:20] <- NA
d <- data.frame(x,y)
f <- aregImpute(~ x + y, n.impute=10, match='closest', data=d)
# As an aside here is how to create a completed dataset for imputation
# number 3 as fit.mult.impute would do automatically. In this degenerate
# case changing 3 to 1-2,4-10 will not alter the results.
imputed <- impute.transcan(f, imputation=3, data=d, list.out=TRUE,
                          pr=FALSE, check=FALSE)
sd <- sqrt(mean(apply(f$imputed$y, 1, var)))

ss <- c(0, .01, .02, seq(.05, 1, length=20))
sds <- ss; sds[1] <- sd

for(i in 2:length(ss)) {
  f <- aregImpute(~ x + y, n.impute=10, fweighted=ss[i])
  sds[i] <- sqrt(mean(apply(f$imputed$y, 1, var)))
}

plot(ss, sds, xlab='Smoothing Parameter', ylab='SD of Imputed Values',
      type='b')
abline(v=.2, lty=2) # default value of fweighted
abline(h=rmse, lty=2) # root MSE of residuals from linear regression

## End(Not run)

## Not run:
# Do a similar experiment for the Titanic dataset
getHdata(titanic3)
h <- lm(age ~ sex + pclass + survived, data=titanic3)
rmse <- summary(h)$sigma
set.seed(21)
f <- aregImpute(~ age + sex + pclass + survived, n.impute=10,
               data=titanic3, match='closest')
sd <- sqrt(mean(apply(f$imputed$age, 1, var)))

ss <- c(0, .01, .02, seq(.05, 1, length=20))
sds <- ss; sds[1] <- sd

for(i in 2:length(ss)) {
  f <- aregImpute(~ age + sex + pclass + survived, data=titanic3,
                 n.impute=10, fweighted=ss[i])
  sds[i] <- sqrt(mean(apply(f$imputed$age, 1, var)))
}

plot(ss, sds, xlab='Smoothing Parameter', ylab='SD of Imputed Values',
      type='b')
abline(v=.2, lty=2) # default value of fweighted

```

```

abline(h=rmse, lty=2) # root MSE of residuals from linear regression

## End(Not run)

set.seed(2)
d <- data.frame(x1=runif(50), x2=c(rep(NA, 10), runif(40)),
                x3=c(runif(4), rep(NA, 11), runif(35)))
reformM(~ x1 + x2 + x3, data=d)
reformM(~ x1 + x2 + x3, data=d, nperm=2)
# Give result or one of the results as the first argument to aregImpute

# Constrain imputed values for two variables
# Require imputed values for x2 to be above 0.2
# Assume x1 is never missing and require imputed values for
# x3 to be less than the recipient's value of x1
a <- aregImpute(~ x1 + x2 + x3, data=d,
                constraint=list(x2 = expression(d$x2 > 0.2),
                               x3 = expression(d$x3 < r$x1)))

a

```

---

binconf

*Confidence Intervals for Binomial Probabilities*


---

## Description

Produces 1-alpha confidence intervals for binomial probabilities.

## Usage

```

binconf(x, n, alpha=0.05,
        method=c("wilson", "exact", "asymptotic", "all"),
        include.x=FALSE, include.n=FALSE, return.df=FALSE)

```

## Arguments

x	vector containing the number of "successes" for binomial variates
n	vector containing the numbers of corresponding observations
alpha	probability of a type I error, so confidence coefficient = 1-alpha
method	character string specifying which method to use. The "all" method only works when x and n are length 1. The "exact" method uses the F distribution to compute exact (based on the binomial cdf) intervals; the "wilson" interval is score-test-based; and the "asymptotic" is the text-book, asymptotic normal interval. Following Agresti and Coull, the Wilson interval is to be preferred and so is the default.
include.x	logical flag to indicate whether x should be included in the returned matrix or data frame

include.n	logical flag to indicate whether n should be included in the returned matrix or data frame
return.df	logical flag to indicate that a data frame rather than a matrix be returned

**Value**

a matrix or data.frame containing the computed intervals and, optionally, x and n.

**Author(s)**

Rollin Brant, Modified by Frank Harrell and  
 Brad Biggerstaff  
 Centers for Disease Control and Prevention  
 National Center for Infectious Diseases  
 Division of Vector-Borne Infectious Diseases  
 P.O. Box 2087, Fort Collins, CO, 80522-2087, USA  
 <bkb5@cdc.gov>

**References**

- A. Agresti and B.A. Coull, Approximate is better than "exact" for interval estimation of binomial proportions, *American Statistician*, **52**:119–126, 1998.
- R.G. Newcombe, Logit confidence intervals and the inverse sinh transformation, *American Statistician*, **55**:200–202, 2001.
- L.D. Brown, T.T. Cai and A. DasGupta, Interval estimation for a binomial proportion (with discussion), *Statistical Science*, **16**:101–133, 2001.

**Examples**

```
binconf(0:10,10,include.x=TRUE,include.n=TRUE)
binconf(46,50,method="all")
```

---

 biVar

*Bivariate Summaries Computed Separately by a Series of Predictors*


---

**Description**

biVar is a generic function that accepts a formula and usual data, subset, and na.action parameters plus a list statinfo that specifies a function of two variables to compute along with information about labeling results for printing and plotting. The function is called separately with each right hand side variable and the same left hand variable. The result is a matrix of bivariate statistics and the statinfo list that drives printing and plotting. The plot method draws a dot plot with x-axis values by default sorted in order of one of the statistics computed by the function.

spearman2 computes the square of Spearman's rho rank correlation and a generalization of it in which x can relate non-monotonically to y. This is done by computing the Spearman multiple rho-squared between (rank(x), rank(x)^2) and y. When x is categorical, a different kind of Spearman correlation used in the Kruskal-Wallis test is computed (and spearman2 can do the Kruskal-Wallis test). This is done by computing the ordinary multiple R^2 between k-1 dummy variables

and `rank(y)`, where `x` has `k` categories. `x` can also be a formula, in which case each predictor is correlated separately with `y`, using non-missing observations for that predictor. `biVar` is used to do the looping and bookkeeping. By default the plot shows the adjusted  $\rho^2$ , using the same formula used for the ordinary adjusted  $R^2$ . The F test uses the unadjusted  $R^2$ .

`spearman` computes Spearman's rho on non-missing values of two variables. `spearman.test` is a simple version of `spearman2.default`.

`chiSquare` is set up like `spearman2` except it is intended for a categorical response variable. Separate Pearson chi-square tests are done for each predictor, with optional collapsing of infrequent categories. Numeric predictors having more than `g` levels are categorized into `g` quantile groups. `chiSquare` uses `biVar`.

## Usage

```
biVar(formula, statinfo, data=NULL, subset=NULL,
      na.action=na.retain, exclude.imputed=TRUE, ...)

## S3 method for class 'biVar'
print(x, ...)

## S3 method for class 'biVar'
plot(x, what=info$defaultwhat,
      sort.=TRUE, main, xlab,
      vnames=c('names','labels'), ...)

spearman2(x, ...)

## Default S3 method:
spearman2(x, y, p=1, minlev=0, na.rm=TRUE, exclude.imputed=na.rm, ...)

## S3 method for class 'formula'
spearman2(formula, data=NULL,
          subset, na.action=na.retain, exclude.imputed=TRUE, ...)

spearman(x, y)

spearman.test(x, y, p=1)

chiSquare(formula, data=NULL, subset=NULL, na.action=na.retain,
          exclude.imputed=TRUE, ...)
```

## Arguments

<code>formula</code>	a formula with a single left side variable
<code>statinfo</code>	see <code>spearman2.formula</code> or <code>chiSquare</code> code
<code>data, subset, na.action</code>	the usual options for models. Default for <code>na.action</code> is to retain all values, NA or not, so that NAs can be deleted in only a pairwise fashion.

<code>exclude.imputed</code>	set to FALSE to include imputed values (created by <code>impute</code> ) in the calculations.
<code>...</code>	other arguments that are passed to the function used to compute the bivariate statistics or to <code>dotchart3</code> for plot.
<code>na.rm</code>	logical; delete NA values?
<code>x</code>	a numeric matrix with at least 5 rows and at least 2 columns (if <code>y</code> is absent). For <code>spearman2</code> , the first argument may be a vector of any type, including character or factor. The first argument may also be a formula, in which case all predictors are correlated individually with the response variable. <code>x</code> may be a formula for <code>spearman2</code> in which case <code>spearman2.formula</code> is invoked. Each predictor in the right hand side of the formula is separately correlated with the response variable. For <code>print</code> or <code>plot</code> , <code>x</code> is an object produced by <code>biVar</code> . For <code>spearman</code> and <code>spearman.test</code> <code>x</code> is a numeric vector, as is <code>y</code> . For <code>chiSquare</code> , <code>x</code> is a formula.
<code>y</code>	a numeric vector
<code>p</code>	for numeric variables, specifies the order of the Spearman $\rho^2$ to use. The default is <code>p=1</code> to compute the ordinary $\rho^2$ . Use <code>p=2</code> to compute the quadratic rank generalization to allow non-monotonicity. <code>p</code> is ignored for categorical predictors.
<code>minlev</code>	minimum relative frequency that a level of a categorical predictor should have before it is pooled with other categories (see <code>combine.levels</code> ) in <code>spearman2</code> and <code>chiSquare</code> (in which case it also applies to the response). The default, <code>minlev=0</code> causes no pooling.
<code>what</code>	specifies which statistic to plot. Possibilities include the column names that appear with the <code>print</code> method is used.
<code>sort.</code>	set <code>sort.=FALSE</code> to suppress sorting variables by the statistic being plotted
<code>main</code>	main title for plot. Default title shows the name of the response variable.
<code>xlab</code>	x-axis label. Default constructed from <code>what</code> .
<code>vnames</code>	set to <code>"labels"</code> to use variable labels in place of names for plotting. If a variable does not have a label the name is always used.

### Details

Uses midranks in case of ties, as described by Hollander and Wolfe. P-values for Spearman, Wilcoxon, or Kruskal-Wallis tests are approximated by using the `t` or `F` distributions.

### Value

`spearman2.default` (the function that is called for a single `x`, i.e., when there is no formula) returns a vector of statistics for the variable. `biVar`, `spearman2.formula`, and `chiSquare` return a matrix with rows corresponding to predictors.

### Author(s)

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
[<fh@fharrell.com>](mailto:fh@fharrell.com)

**References**

Hollander M. and Wolfe D.A. (1973). Nonparametric Statistical Methods. New York: Wiley.  
 Press WH, Flannery BP, Teukolsky SA, Vetterling, WT (1988): Numerical Recipes in C. Cambridge: Cambridge University Press.

**See Also**

[combine.levels](#), [varclus](#), [dotchart3](#), [impute](#), [chisq.test](#), [cut2](#).

**Examples**

```
x <- c(-2, -1, 0, 1, 2)
y <- c(4, 1, 0, 1, 4)
z <- c(1, 2, 3, 4, NA)
v <- c(1, 2, 3, 4, 5)

spearman2(x, y)
plot(spearman2(z ~ x + y + v, p=2))

f <- chiSquare(z ~ x + y + v)
f
```

---

 bootkm

*Bootstrap Kaplan-Meier Estimates*


---

**Description**

Bootstraps Kaplan-Meier estimate of the probability of survival to at least a fixed time (times variable) or the estimate of the q quantile of the survival distribution (e.g., median survival time, the default).

**Usage**

```
bootkm(S, q=0.5, B=500, times, pr=TRUE)
```

**Arguments**

S	a Surv object for possibly right-censored survival time
q	quantile of survival time, default is 0.5 for median
B	number of bootstrap repetitions (default=500)
times	time vector (currently only a scalar is allowed) at which to compute survival estimates. You may specify only one of q and times, and if times is specified q is ignored.
pr	set to FALSE to suppress printing the iteration number every 10 iterations

**Details**

bootkm uses Therneau's `survfitKM` function to efficiently compute Kaplan-Meier estimates.

**Value**

a vector containing B bootstrap estimates

**Side Effects**

updates `.Random.seed`, and, if `pr=TRUE`, prints progress of simulations

**Author(s)**

Frank Harrell  
Department of Biostatistics  
Vanderbilt University School of Medicine  
<fh@fharrell.com>

**References**

Akritis MG (1986): Bootstrapping the Kaplan-Meier estimator. *JASA* 81:1032–1038.

**See Also**

[survfit](#), [Surv](#), [Survival.cph](#), [Quantile.cph](#)

**Examples**

```
# Compute 0.95 nonparametric confidence interval for the difference in
# median survival time between females and males (two-sample problem)
set.seed(1)
library(survival)
S <- Surv(runif(200))      # no censoring
sex <- c(rep('female',100),rep('male',100))
med.female <- bootkm(S[sex=='female'], B=100) # normally B=500
med.male   <- bootkm(S[sex=='male'], B=100)
describe(med.female-med.male)
quantile(med.female-med.male, c(.025,.975), na.rm=TRUE)
# na.rm needed because some bootstrap estimates of median survival
# time may be missing when a bootstrap sample did not include the
# longer survival times
```

**Description**

Uses method of Fleiss, Tytun, and Ury (but without the continuity correction) to estimate the power (or the sample size to achieve a given power) of a two-sided test for the difference in two proportions. The two sample sizes are allowed to be unequal, but for `bsamsize` you must specify the fraction of observations in group 1. For power calculations, one probability (`p1`) must be given, and either the other probability (`p2`), an `odds.ratio`, or a `percent.reduction` must be given. For `bpower` or `bsamsize`, any or all of the arguments may be vectors, in which case they return a vector of powers or sample sizes. All vector arguments must have the same length.

Given `p1`, `p2`, `ballocation` uses the method of Brittain and Schlesselman to compute the optimal fraction of observations to be placed in group 1 that either (1) minimize the variance of the difference in two proportions, (2) minimize the variance of the ratio of the two proportions, (3) minimize the variance of the log odds ratio, or (4) maximize the power of the 2-tailed test for differences. For (4) the total sample size must be given, or the fraction optimizing the power is not returned. The fraction for (3) is one minus the fraction for (1).

`bpower.sim` estimates power by simulations, in minimal time. By using `bpower.sim` you can see that the formulas without any continuity correction are quite accurate, and that the power of a continuity-corrected test is significantly lower. That's why no continuity corrections are implemented here.

**Usage**

```
bpower(p1, p2, odds.ratio, percent.reduction,
       n, n1, n2, alpha=0.05)
```

```
bsamsize(p1, p2, fraction=.5, alpha=.05, power=.8)
```

```
ballocation(p1, p2, n, alpha=.05)
```

```
bpower.sim(p1, p2, odds.ratio, percent.reduction,
           n, n1, n2,
           alpha=0.05, nsim=10000)
```

**Arguments**

<code>p1</code>	population probability in the group 1
<code>p2</code>	probability for group 2
<code>odds.ratio</code>	odds ratio to detect
<code>percent.reduction</code>	percent reduction in risk to detect



n	total sample size over the two groups. If you omit this for ballocation, the fraction which optimizes power will not be returned.
n1	sample size in group 1
n2	sample size in group 2. bpower, if n is given, n1 and n2 are set to n/2.
alpha	type I assertion probability
fraction	fraction of observations in group 1
power	the desired probability of detecting a difference
nsim	number of simulations of binomial responses

### Details

For `bpower.sim`, all arguments must be of length one.

### Value

for `bpower`, the power estimate; for `bsamsize`, a vector containing the sample sizes in the two groups; for `ballocation`, a vector with 4 fractions of observations allocated to group 1, optimizing the four criteria mentioned above. For `bpower.sim`, a vector with three elements is returned, corresponding to the simulated power and its lower and upper 0.95 confidence limits.

### AUTHOR

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>

### References

- Fleiss JL, Tytun A, Ury HK (1980): A simple approximation for calculating sample sizes for comparing independent proportions. *Biometrics* 36:343–6.
- Brittain E, Schlesselman JJ (1982): Optimal allocation for the comparison of proportions. *Biometrics* 38:1003–9.
- Gordon I, Watson R (1996): The myth of continuity-corrected sample size formulae. *Biometrics* 52:71–6.

### See Also

[samplesize.bin](#), [chisq.test](#), [binconf](#)

### Examples

```
bpower(.1, odds.ratio=.9, n=1000, alpha=c(.01,.05))
bpower.sim(.1, odds.ratio=.9, n=1000)
bsamsize(.1, .05, power=.95)
ballocation(.1, .5, n=100)
```

```

# Plot power vs. n for various odds ratios (base prob.=.1)
n <- seq(10, 1000, by=10)
OR <- seq(.2,.9,by=.1)
plot(0, 0, xlim=range(n), ylim=c(0,1), xlab="n", ylab="Power", type="n")
for(or in OR) {
  lines(n, bpower(.1, odds.ratio=or, n=n))
  text(350, bpower(.1, odds.ratio=or, n=350)-.02, format(or))
}

# Another way to plot the same curves, but letting labcurve do the
# work, including labeling each curve at points of maximum separation
pow <- lapply(OR, function(or,n)list(x=n,y=bpower(p1=.1,odds.ratio=or,n=n)),
             n=n)
names(pow) <- format(OR)
labcurve(pow, pl=TRUE, xlab='n', ylab='Power')

# Contour graph for various probabilities of outcome in the control
# group, fixing the odds ratio at .8 ( $[p2/(1-p2) / p1/(1-p1)] = .8$ )
# n is varied also
p1 <- seq(.01,.99,by=.01)
n <- seq(100,5000,by=250)
pow <- outer(p1, n, function(p1,n) bpower(p1, n=n, odds.ratio=.8))
# This forms a length(p1)*length(n) matrix of power estimates
contour(p1, n, pow)

```

---

bpplot

*Box-percentile plots*


---

## Description

Produce side-by-side box-percentile plots from several vectors or a list of vectors.

## Usage

```
bpplot(..., name=TRUE, main="Box-Percentile Plot",
       xlab="", ylab="", srtx=0, plotopts=NULL)
```

## Arguments

...	vectors or lists containing numeric components (e.g., the output of <code>split</code> ).
name	character vector of names for the groups. Default is TRUE to put names on the x-axis. Such names are taken from the data vectors or the names attribute of the first argument if it is a list. Set name to FALSE to suppress names. If a character vector is supplied the names in the vector are used to label the groups.
main	main title for the plot.
xlab	x axis label.

ylab	y axis label.
srtx	rotation angle for x-axis labels. Default is zero.
plotopts	a list of other parameters to send to plot

**Value**

There are no returned values

**Side Effects**

A plot is created on the current graphics device.

**BACKGROUND**

Box-percentile plots are similar to boxplots, except box-percentile plots supply more information about the univariate distributions. At any height the width of the irregular "box" is proportional to the percentile of that height, up to the 50th percentile, and above the 50th percentile the width is proportional to 100 minus the percentile. Thus, the width at any given height is proportional to the percent of observations that are more extreme in that direction. As in boxplots, the median, 25th and 75th percentiles are marked with line segments across the box.

**Author(s)**

Jeffrey Banfield  
<umsfjban@bill.oscs.montana.edu>  
Modified by F. Harrell 30Jun97

**References**

Esty WW, Banfield J: The box-percentile plot. J Statistical Software 8 No. 17, 2003.

**See Also**

[panel.bpplot](#), [boxplot](#), [Ecdf](#), [bwplot](#)

**Examples**

```
set.seed(1)
x1 <- rnorm(500)
x2 <- runif(500, -2, 2)
x3 <- abs(rnorm(500))-2
bpplot(x1, x2, x3)
g <- sample(1:2, 500, replace=TRUE)
bpplot(split(x2, g), name=c('Group 1', 'Group 2'))
rm(x1,x2,x3,g)
```

## Description

For any number of cross-classification variables, `bystats` returns a matrix with the sample size, number missing  $y$ , and `fun`(non-missing  $y$ ), with the cross-classifications designated by rows. Uses Harrell's modification of the interaction function to produce cross-classifications. The default `fun` is `mean`, and if  $y$  is binary, the mean is labeled as `Fraction`. There is a `print` method as well as a `latex` method for objects created by `bystats`. `bystats2` handles the special case in which there are 2 classification variables, and places the first one in rows and the second in columns. The `print` method for `bystats2` uses the `print.char.matrix` function to organize statistics for cells into boxes.

## Usage

```
bystats(y, ..., fun, nmiss, subset)
## S3 method for class 'bystats'
print(x, ...)
## S3 method for class 'bystats'
latex(object, title, caption, rowlabel, ...)
bystats2(y, v, h, fun, nmiss, subset)
## S3 method for class 'bystats2'
print(x, abbreviate.dimnames=FALSE,
      prefix.width=max(nchar(dimnames(x)[[1]])), ...)
## S3 method for class 'bystats2'
latex(object, title, caption, rowlabel, ...)
```

## Arguments

- |                  |  |
|------------------|--|
| <code>y</code>   | a binary, logical, or continuous variable or a matrix or data frame of such variables. If $y$ is a data frame it is converted to a matrix. If $y$ is a data frame or matrix, computations are done on subsets of the rows of $y$ , and you should specify <code>fun</code> so as to be able to operate on the matrix. For matrix $y$ , any column with a missing value causes the entire row to be considered missing, and the row is not passed to <code>fun</code> .   |
| <code>...</code> | For <code>bystats</code> , one or more classification variables separated by commas. For <code>print.bystats</code> , options passed to <code>print.default</code> such as <code>digits</code> . For <code>latex.bystats</code> , and <code>latex.bystats2</code> , options passed to <code>latex.default</code> such as <code>digits</code> . If you pass <code>cdec</code> to <code>latex.default</code> , keep in mind that the first one or two positions (depending on <code>nmiss</code> ) should have zeros since these correspond with frequency counts. |
| <code>v</code>   | vertical variable for <code>bystats2</code> . Will be converted to factor.   |
| <code>h</code>   | horizontal variable for <code>bystats2</code> . Will be converted to factor.   |

fun	a function to compute on the non-missing y for a given subset. You must specify fun= in front of the function name or definition. fun may return a single number or a vector or matrix of any length. Matrix results are rolled out into a vector, with names preserved. When y is a matrix, a common fun is function(y) apply(y, 2, ff) where ff is the name of a function which operates on one column of y.
nmiss	A column containing a count of missing values is included if nmiss=TRUE or if there is at least one missing value.
subset	a vector of subscripts or logical values indicating the subset of data to analyze
abbreviate.dimnames	set to TRUE to abbreviate dimnames in output
prefix.width	see <a href="#">print.char.matrix</a>
title	title to pass to latex.default. Default is the first word of the character string version of the first calling argument.
caption	caption to pass to latex.default. Default is the heading attribute from the object produced by <code>bystats</code> .
rowlabel	rowlabel to pass to latex.default. Default is the byvarnames attribute from the object produced by <code>bystats</code> . For <code>bystats2</code> the default is "".
x	an object created by <code>bystats</code> or <code>bystats2</code>
object	an object created by <code>bystats</code> or <code>bystats2</code>

**Value**

for `bystats`, a matrix with row names equal to the classification labels and column names N, Missing, funlab, where funlab is determined from fun. A row is added to the end with the summary statistics computed on all observations combined. The class of this matrix is `bystats`. For `bystats2`, returns a 3-dimensional array with the last dimension corresponding to statistics being computed. The class of the array is `bystats2`.

**Side Effects**

latex produces a .tex file.

**Author(s)**

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>

**See Also**

[interaction](#), [cut](#), [cut2](#), [latex](#), [print.char.matrix](#), [translate](#)

**Examples**

```
## Not run:
bystats(sex==2, county, city)
bystats(death, race)
bystats(death, cut2(age,g=5), race)
bystats(cholesterol, cut2(age,g=4), sex, fun=median)
bystats(cholesterol, sex, fun=quantile)
bystats(cholesterol, sex, fun=function(x)c(Mean=mean(x),Median=median(x)))
latex(bystats(death,race,nmiss=FALSE,subset=sex=="female"), digits=2)
f <- function(y) c(Hazard=sum(y[,2])/sum(y[,1]))
# f() gets the hazard estimate for right-censored data from exponential dist.
bystats(cbind(d.time, death), race, sex, fun=f)
bystats(cbind(pressure, cholesterol), age.decile,
        fun=function(y) c(Median.pressure =median(y[,1]),
                          Median.cholesterol=median(y[,2])))
y <- cbind(pressure, cholesterol)
bystats(y, age.decile,
        fun=function(y) apply(y, 2, median)) # same result as last one
bystats(y, age.decile, fun=function(y) apply(y, 2, quantile, c(.25,.75)))
# The last one computes separately the 0.25 and 0.75 quantiles of 2 vars.
latex(bystats2(death, race, sex, fun=table))

## End(Not run)
```

---

capitalize

*capitalize the first letter of a string*

---

**Description**

Capitalizes the first letter of each element of the string vector.

**Usage**

```
capitalize(string)
```

**Arguments**

string           String to be capitalized

**Value**

Returns a vector of characters with the first letter capitalized

**Author(s)**

Charles Dupont

**Examples**

```
capitalize(c("Hello", "bob", "daN"))
```

---

`ciapower`*Power of Interaction Test for Exponential Survival*

---

**Description**

Uses the method of Peterson and George to compute the power of an interaction test in a 2 x 2 setup in which all 4 distributions are exponential. This will be the same as the power of the Cox model test if assumptions hold. The test is 2-tailed. The duration of accrual is specified (constant accrual is assumed), as is the minimum follow-up time. The maximum follow-up time is then accrual + tmin. Treatment allocation is assumed to be 1:1.

**Usage**

```
ciapower(tref, n1, n2, m1c, m2c, r1, r2, accrual, tmin,  
         alpha=0.05, pr=TRUE)
```

**Arguments**

tref	time at which mortalities estimated
n1	total sample size, stratum 1
n2	total sample size, stratum 2
m1c	tref-year mortality, stratum 1 control
m2c	tref-year mortality, stratum 2 control
r1	% reduction in m1c by intervention, stratum 1
r2	% reduction in m2c by intervention, stratum 2
accrual	duration of accrual period
tmin	minimum follow-up time
alpha	type I error probability
pr	set to FALSE to suppress printing of details

**Value**

power

**Side Effects**

prints

**AUTHOR**

Frank Harrell  
Department of Biostatistics  
Vanderbilt University  
<fh@fharrell.com>

**References**

Peterson B, George SL: Controlled Clinical Trials 14:511–522; 1993.

**See Also**

[cpower](#), [spower](#)

**Examples**

```
# Find the power of a race x treatment test. 25% of patients will
# be non-white and the total sample size is 14000.
# Accrual is for 1.5 years and minimum follow-up is 5y.
# Reduction in 5-year mortality is 15% for whites, 0% or -5% for
# non-whites. 5-year mortality for control subjects if assumed to
# be 0.18 for whites, 0.23 for non-whites.
n <- 14000
for(nonwhite.reduction in c(0,-5)) {
  cat("\n\n\n% Reduction in 5-year mortality for non-whites:",
      nonwhite.reduction, "\n\n")
  pow <- ciapower(5, .75*n, .25*n, .18, .23, 15, nonwhite.reduction,
                 1.5, 5)
  cat("\n\nPower:", format(pow), "\n")
}
```

---

cnvrt.coords

*Convert between the 5 different coordinate systems on a graphical device*

---

**Description**

Takes a set of coordinates in any of the 5 coordinate systems (usr, plt, fig, dev, or tdev) and returns the same points in all 5 coordinate systems.

**Usage**

```
cnvrt.coords(x, y = NULL, input = c("usr", "plt", "fig", "dev", "tdev"))
```

**Arguments**

**x** Vector, Matrix, or list of x coordinates (or x and y coordinates), NA's allowed.

**y** y coordinates (if x is a vector), NA's allowed.

**input** Character scalar indicating the coordinate system of the input points.



**Details**

Every plot has 5 coordinate systems:

`usr` (User): the coordinate system of the data, this is shown by the tick marks and axis labels.

`plt` (Plot): Plot area, coordinates range from 0 to 1 with 0 corresponding to the x and y axes and 1 corresponding to the top and right of the plot area. Margins of the plot correspond to plot coordinates less than 0 or greater than 1.

`fig` (Figure): Figure area, coordinates range from 0 to 1 with 0 corresponding to the bottom and left edges of the figure (including margins, label areas) and 1 corresponds to the top and right edges. `fig` and `dev` coordinates will be identical if there is only 1 figure area on the device (layout, `mfrow`, or `mfc` has not been used).

`dev` (Device): Device area, coordinates range from 0 to 1 with 0 corresponding to the bottom and left of the device region within the outer margins and 1 is the top and right of the region within the outer margins. If the outer margins are all set to 0 then `tdev` and `dev` should be identical.

`tdev` (Total Device): Total Device area, coordinates range from 0 to 1 with 0 corresponding to the bottom and left edges of the device (piece of paper, window on screen) and 1 corresponds to the top and right edges.

**Value**

A list with 5 components, each component is a list with vectors named `x` and `y`. The 5 sublists are:

<code>usr</code>	The coordinates of the input points in <code>usr</code> (User) coordinates.
<code>plt</code>	The coordinates of the input points in <code>plt</code> (Plot) coordinates.
<code>fig</code>	The coordinates of the input points in <code>fig</code> (Figure) coordinates.
<code>dev</code>	The coordinates of the input points in <code>dev</code> (Device) coordinates.
<code>tdev</code>	The coordinates of the input points in <code>tdev</code> (Total Device) coordinates.

**Note**

You must provide both `x` and `y`, but one of them may be `NA`.

This function is becoming deprecated with the new functions `grconvertX` and `grconvertY` in R version 2.7.0 and beyond. These new functions use the correct coordinate system names and have more coordinate systems available, you should start using them instead.

**Author(s)**

Greg Snow <greg.snow@imail.org>

**See Also**

`par` specifically `'usr'`, `'plt'`, and `'fig'`. Also `'xpd'` for plotting outside of the plotting region and `'mfrow'` and `'mfc'` for multi figure plotting. `subplot`, `grconvertX` and `grconvertY` in R2.7.0 and later

**Examples**

```

old.par <- par(no.readonly=TRUE)

par(mfrow=c(2,2),xpd=NA)

# generate some sample data
tmp.x <- rnorm(25, 10, 2)
tmp.y <- rnorm(25, 50, 10)
tmp.z <- rnorm(25, 0, 1)

plot( tmp.x, tmp.y)

# draw a diagonal line across the plot area
tmp1 <- cnvrt.coords( c(0,1), c(0,1), input='plt' )
lines(tmp1$usr, col='blue')

# draw a diagonal line accross figure region
tmp2 <- cnvrt.coords( c(0,1), c(1,0), input='fig')
lines(tmp2$usr, col='red')

# save coordinate of point 1 and y value near top of plot for future plots
tmp.point1 <- cnvrt.coords(tmp.x[1], tmp.y[1])
tmp.range1 <- cnvrt.coords(NA, 0.98, input='plt')

# make a second plot and draw a line linking point 1 in each plot
plot(tmp.y, tmp.z)

tmp.point2 <- cnvrt.coords( tmp.point1$dev, input='dev' )
arrows( tmp.y[1], tmp.z[1], tmp.point2$usr$x, tmp.point2$usr$y,
        col='green')

# draw another plot and add rectangle showing same range in 2 plots

plot(tmp.x, tmp.z)
tmp.range2 <- cnvrt.coords(NA, 0.02, input='plt')
tmp.range3 <- cnvrt.coords(NA, tmp.range1$dev$y, input='dev')
rect( 9, tmp.range2$usr$y, 11, tmp.range3$usr$y, border='yellow')

# put a label just to the right of the plot and
# near the top of the figure region.
text( cnvrt.coords(1.05, NA, input='plt')$usr$x,
      cnvrt.coords(NA, 0.75, input='fig')$usr$y,
      "Label", adj=0)

par(mfrow=c(1,1))

## create a subplot within another plot (see also subplot)

plot(1:10, 1:10)

tmp <- cnvrt.coords( c( 1, 4, 6, 9), c(6, 9, 1, 4) )

```

```

par(p1t = c(tmp$dev$x[1:2], tmp$dev$y[1:2]), new=TRUE)
hist(rnorm(100))

par(fig = c(tmp$dev$x[3:4], tmp$dev$y[3:4]), new=TRUE)
hist(rnorm(100))

par(old.par)

```

---

colorFacet

*Miscellaneous ggplot2 and grid Helper Functions*


---

### Description

These functions are used on ggplot2 objects or as layers when building a ggplot2 object, and to facilitate use of gridExtra. colorFacet colors the thin rectangles used to separate panels created by facet\_grid (and probably by facet\_wrap). A better approach may be found at <https://stackoverflow.com/questions/28652284/>. arrGrob is a front-end to gridExtra::arrangeGrob that allows for proper printing. See <https://stackoverflow.com/questions/29062766/store-output-from-gridextra>. The arrGrob print method calls grid::grid.draw.

### Usage

```

colorFacet(g, col = adjustcolor("blue", alpha.f = 0.3))

arrGrob(...)

## S3 method for class 'arrGrob'
print(x, ...)

```

### Arguments

g	a ggplot2 object that used faceting
col	color for facet separator rectangles
...	passed to arrangeGrob
x	an object created by arrGrob

### Author(s)

Sandy Muspratt

### Examples

```

## Not run:
require(ggplot2)
s <- summaryP(age + sex ~ region + treatment)
colorFacet(ggplot(s)) # prints directly
# arrGrob is called by rms::ggplot.Predict and others

```

```
## End(Not run)
```

---

combine.levels	<i>combine.levels</i>
----------------	-----------------------

---

## Description

Combine Infrequent Levels of a Categorical Variable

## Usage

```
combine.levels(
  x,
  minlev = 0.05,
  m,
  ord = is.ordered(x),
  plevels = FALSE,
  sep = ", "
)
```

## Arguments

x	a factor, ‘ordered’ factor, or numeric or character variable that will be turned into a ‘factor’
minlev	the minimum proportion of observations in a cell before that cell is combined with one or more cells. If more than one cell has fewer than $\text{minlev} \times n$ observations, all such cells are combined into a new cell labeled “OTHER”. Otherwise, the lowest frequency cell is combined with the next lowest frequency cell, and the level name is the combination of the two old level levels. When ‘ord=TRUE’ combinations happen only for consecutive levels.
m	alternative to ‘minlev’, is the minimum number of observations in a cell before it will be combined with others
ord	set to ‘TRUE’ to treat ‘x’ as if it were an ordered factor, which allows only consecutive levels to be combined
plevels	by default ‘combine.levels’ pools low-frequency levels into a category named ‘OTHER’ when ‘x’ is not ordered and ‘ord=FALSE’. To instead name this category the concatenation of all the pooled level names, separated by a comma, set ‘plevels=TRUE’.
sep	the separator for concatenating levels when ‘plevels=TRUE’

**Details**

After turning 'x' into a 'factor' if it is not one already, combines levels of 'x' whose frequency falls below a specified relative frequency 'minlev' or absolute count 'm'. When 'x' is not treated as ordered, all of the small frequency levels are combined into "OTHER", unless 'plevels=TRUE'. When 'ord=TRUE' or 'x' is an ordered factor, only consecutive levels are combined. New levels are constructed by concatenating the levels with 'sep' as a separator. This is useful when comparing ordinal regression with polytomous (multinomial) regression and there are too many categories for polytomous regression. 'combine.levels' is also useful when assumptions of ordinal models are being checked empirically by computing exceedance probabilities for various cutoffs of the dependent variable.

**Value**

a factor variable, or if 'ord=TRUE' an ordered factor variable

**Author(s)**

Frank Harrell

**Examples**

```
x <- c(rep('A', 1), rep('B', 3), rep('C', 4), rep('D',1), rep('E',1))
combine.levels(x, m=3)
combine.levels(x, m=3, plevels=TRUE)
combine.levels(x, ord=TRUE, m=3)
x <- c(rep('A', 1), rep('B', 3), rep('C', 4), rep('D',1), rep('E',1),
      rep('F',1))
combine.levels(x, ord=TRUE, m=3)
```

---

combplotp

*Combination Plot*

---

**Description**

Generates a plotly attribute plot given a series of possibly overlapping binary variables

**Usage**

```
combplotp(
  formula,
  data = NULL,
  subset,
  na.action = na.retain,
  vnames = c("labels", "names"),
  includenone = FALSE,
  showno = FALSE,
  maxcomb = NULL,
  minfreq = NULL,
```

```

N = NULL,
pos = function(x) 1 * (tolower(x) %in% c("true", "yes", "y", "positive", "+",
  "present", "1")),
obsname = "subjects",
ptsize = 35,
width = NULL,
height = NULL,
...
)

```

### Arguments

formula	a formula containing all the variables to be cross-tabulated, on the formula's right hand side. There is no left hand side variable. If formula is omitted, then all variables from data are analyzed.
data	input data frame. If none is specified the data are assumed to come from the parent frame.
subset	an optional subsetting expression applied to data
na.action	see lm etc.
vnames	set to "names" to use variable names to label axes instead of variable labels. When using the default labels, any variable not having a label will have its name used instead.
includenone	set to TRUE to include the combination where all conditions are absent
showno	set to TRUE to show a light dot for conditions that are not part of the currently tabulated combination
maxcomb	maximum number of combinations to display
minfreq	if specified, any combination having a frequency less than this will be omitted from the display
N	set to an integer to override the global denominator, instead of using the number of rows in the data
pos	a function of vector returning a logical vector with TRUE values indicating positive
obsname	character string noun describing observations, default is "subjects"
ptsize	point size, defaults to 35
width	width of plotly plot
height	height of plotly plot
...	optional arguments to pass to table

### Details

Similar to the UpSetR package, draws a special dot chart sometimes called an attribute plot that depicts all possible combination of the binary variables. By default a positive value, indicating that a certain condition pertains for a subject, is any of logical TRUE, numeric 1, "yes", "y", "positive", "+" or "present" value, and all others are considered negative. The user can override this determination by specifying her own pos function. Case is ignored in the variable values.

The plot uses solid dots arranged in a vertical line to indicate which combination of conditions is being considered. Frequencies of all possible combinations are shown above the dot chart. Marginal frequencies of positive values for the input variables are shown to the left of the dot chart. More information for all three of these component symbols is provided in hover text.

Variables are sorted in descending order of marginal frequencies and likewise for combinations of variables.

### Value

plotly object

### Author(s)

Frank Harrell

### Examples

```
if (requireNamespace("plotly")) {
  g <- function() sample(0:1, n, prob=c(1 - p, p), replace=TRUE)
  set.seed(2); n <- 100; p <- 0.5
  x1 <- g(); label(x1) <- 'A long label for x1 that describes it'
  x2 <- g()
  x3 <- g(); label(x3) <- 'This is<br>a label for x3'
  x4 <- g()
  combplotp(~ x1 + x2 + x3 + x4, showno=TRUE, includenone=TRUE)

  n <- 1500; p <- 0.05
  pain <- g()
  anxiety <- g()
  depression <- g()
  soreness <- g()
  numbness <- g()
  tiredness <- g()
  sleepiness <- g()
  combplotp(~ pain + anxiety + depression + soreness + numbness +
            tiredness + sleepiness, showno=TRUE)
}
```

---

completer

*completer*

---

### Description

Create imputed dataset(s) using transcan and aregImpute objects

### Usage

```
completer(a, nimpute, oneimpute = FALSE, mydata)
```

**Arguments**

<code>a</code>	An object of class <code>transcan</code> or <code>aregImpute</code>
<code>nimpute</code>	A numeric vector between 1 and <code>a\$n.impute</code> . For <code>transcan</code> object, this is set to 1. For <code>aregImpute</code> object, returns a list of <code>nimpute</code> datasets when <code>oneimpute</code> is set to <code>FALSE</code> (default).
<code>oneimpute</code>	A logical vector. When set to <code>TRUE</code> , returns a single completed dataset for the imputation number specified by <code>nimpute</code>
<code>mydata</code>	A data frame in which its missing values will be imputed.

**Details**

Similar in function to `mice::complete`, this function uses `transcan` and `aregImpute` objects to impute missing data and returns the completed dataset(s) as a dataframe or a list. It assumes that `transcan` is used for single regression imputation.

**Value**

A single or a list of completed dataset(s).

**Author(s)**

Yong-Hao Pua, Singapore General Hospital

**Examples**

```
## Not run:
mtcars$hp[1:5] <- NA
mtcars$wt[1:10] <- NA
myrform <- ~ wt + hp + I(carb)
mytranscan <- transcan( myrform, data = mtcars, imputed = TRUE,
  pl = FALSE, pr = FALSE, trantab = TRUE, long = TRUE)
myareg <- aregImpute(myrform, data = mtcars, x=TRUE, n.impute = 5)
completer(mytranscan) # single completed dataset
completer(myareg, 3, oneimpute = TRUE)
# single completed dataset based on the `n.impute`th set of multiple imputation
completer(myareg, 3)
# list of completed datasets based on first `nimpute` sets of multiple imputation
completer(myareg)
# list of completed datasets based on all available sets of multiple imputation
# To get a stacked data frame of all completed datasets use
# do.call(rbind, completer(myareg, data=mydata))
# or use rbindlist in data.table

## End(Not run)
```



---

consolidate	<i>Element Merging</i>
-------------	------------------------

---

**Description**

Merges an object by the names of its elements. Inserting elements in value into x that do not exist in x and replacing elements in x that exist in value with value elements if protect is false.

**Usage**

```
consolidate(x, value, protect, ...)
## Default S3 method:
consolidate(x, value, protect=FALSE, ...)

consolidate(x, protect, ...) <- value
```

**Arguments**

x	named list or vector
value	named list or vector
protect	logical; should elements in x be kept instead of elements in value?
...	currently does nothing; included if ever want to make generic.

**Author(s)**

Charles Dupont

**See Also**

[names](#)

**Examples**

```
x <- 1:5
names(x) <- LETTERS[x]

y <- 6:10
names(y) <- LETTERS[y-2]

x           # c(A=1,B=2,C=3,D=4,E=5)
y           # c(D=6,E=7,F=8,G=9,H=10)

consolidate(x, y)      # c(A=1,B=2,C=3,D=6,E=7,F=8,G=9,H=10)
consolidate(x, y, protect=TRUE) # c(A=1,B=2,C=3,D=4,E=5,F=8,G=9,H=10)
```

**Description**

contents is a generic method for which contents.data.frame is currently the only method. contents.data.frame creates an object containing the following attributes of the variables from a data frame: names, labels (if any), units (if any), number of factor levels (if any), factor levels, class, storage mode, and number of NAs. print.contents.data.frame will print the results, with options for sorting the variables. html.contents.data.frame creates HTML code for displaying the results. This code has hyperlinks so that if the user clicks on the number of levels the browser jumps to the correct part of a table of factor levels for all the factor variables. If long labels are present ("longlabel" attributes on variables), these are printed at the bottom and the html method links to them through the regular labels. Variables having the same levels in the same order have the levels factored out for brevity.

contents.list prints a directory of datasets when [sasxport.get](#) imported more than one SAS dataset.

If options(prType='html') is in effect, calling print on an object that is the contents of a data frame will result in rendering the HTML version. If run from the console a browser window will open.

**Usage**

```
contents(object, ...)
## S3 method for class 'data.frame'
contents(object, sortlevels=FALSE, id=NULL,
  range=NULL, values=NULL, ...)
## S3 method for class 'contents.data.frame'
print(x,
  sort=c('none','names','labels','NAs'), prlevels=TRUE, maxlevels=Inf,
  number=FALSE, ...)
## S3 method for class 'contents.data.frame'
html(object,
  sort=c('none','names','labels','NAs'), prlevels=TRUE, maxlevels=Inf,
  levelType=c('list','table'),
  number=FALSE, nshow=TRUE, ...)
## S3 method for class 'list'
contents(object, dslabels, ...)
## S3 method for class 'contents.list'
print(x,
  sort=c('none','names','labels','NAs','vars'), ...)
```

**Arguments**

object            a data frame. For html is an object created by contents. For contents.list is a list of data frames.

sortlevels	set to TRUE to sort levels of all factor variables into alphabetic order. This is especially useful when two variables use the same levels but in different orders. They will still be recognized by the <code>html</code> method as having identical levels if sorted.
id	an optional subject ID variable name that if present in object will cause the number of unique IDs to be printed in the contents header
range	an optional variable name that if present in object will cause its range to be printed in the contents header
values	an optional variable name that if present in object will cause its unique values to be printed in the contents header
x	an object created by contents
sort	Default is to print the variables in their original order in the data frame. Specify one of "names", "labels", or "NAs" to sort the variables by, respectively, alphabetically by names, alphabetically by labels, or by increasing order of number of missing values. For <code>contents.list</code> , <code>sort</code> may also be the value "vars" to cause sorting by the number of variables in the dataset.
prlevels	set to FALSE to not print all levels of factor variables
maxlevels	maximum number of levels to print for a factor variable
number	set to TRUE to have the <code>print</code> and <code>latex</code> methods number the variables by their order in the data frame
nshow	set to FALSE to suppress outputting number of observations and number of NAs; useful when these counts would unblind information to blinded reviewers
levelType	By default, bullet lists of category levels are constructed in html. Set <code>levelType= 'table'</code> to put levels in html table format.
...	arguments passed from <code>html</code> to <code>format.df</code> , unused otherwise
dslabels	named vector of SAS dataset labels, created for example by <a href="#">sasdsLabels</a>

**Value**

an object of class "contents.data.frame" or "contents.list". For the `html` method is an html character vector object.

**Author(s)**

Frank Harrell  
 Vanderbilt University  
 <fh@fharrell.com>

**See Also**

[describe](#), [html](#), [upData](#), [extractlabs](#), [hlab](#)

**Examples**

```

set.seed(1)
dfr <- data.frame(x=rnorm(400),y=sample(c('male','female'),400,TRUE),
                 stringsAsFactors=TRUE)

contents(dfr)
dfr <- upData(dfr, labels=c(x='Label for x', y='Label for y'))
attr(dfr$x, 'longlabel') <-
  'A very long label for x that can continue onto multiple long lines of text'

k <- contents(dfr)
print(k, sort='names', prlevels=FALSE)
## Not run:
html(k)
html(contents(dfr))          # same result
latex(k$contents)          # latex.default just the main information

## End(Not run)

```

---

cpower

---

*Power of Cox/log-rank Two-Sample Test*


---

**Description**

Assumes exponential distributions for both treatment groups. Uses the George-Desu method along with formulas of Schoenfeld that allow estimation of the expected number of events in the two groups. To allow for drop-ins (noncompliance to control therapy, crossover to intervention) and noncompliance of the intervention, the method of Lachin and Foulkes is used.

**Usage**

```

cpower(tref, n, mc, r, accrual, tmin, noncomp.c=0, noncomp.i=0,
       alpha=0.05, nc, ni, pr=TRUE)

```

**Arguments**

tref	time at which mortalities estimated
n	total sample size (both groups combined). If allocation is unequal so that there are not $n/2$ observations in each group, you may specify the sample sizes in <code>nc</code> and <code>ni</code> .
mc	tref-year mortality, control
r	% reduction in <code>mc</code> by intervention
accrual	duration of accrual period
tmin	minimum follow-up time
noncomp.c	% non-compliant in control group (drop-ins)
noncomp.i	% non-compliant in intervention group (non-adherers)

alpha	type I error probability. A 2-tailed test is assumed.
nc	number of subjects in control group
ni	number of subjects in intervention group. nc and ni are specified exclusive of n.
pr	set to FALSE to suppress printing of details

### Details

For handling noncompliance, uses a modification of formula (5.4) of Lachin and Foulkes. Their method is based on a test for the difference in two hazard rates, whereas cpower is based on testing the difference in two log hazards. It is assumed here that the same correction factor can be approximately applied to the log hazard ratio as Lachin and Foulkes applied to the hazard difference.

Note that Schoenfeld approximates the variance of the log hazard ratio by  $4/m$ , where  $m$  is the total number of events, whereas the George-Desu method uses the slightly better  $1/m_1 + 1/m_2$ . Power from this function will thus differ slightly from that obtained with the SAS `samsizc` program.

### Value

power

### Side Effects

prints

### Author(s)

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>

### References

Peterson B, George SL: *Controlled Clinical Trials* 14:511–522; 1993.  
 Lachin JM, Foulkes MA: *Biometrics* 42:507–519; 1986.  
 Schoenfeld D: *Biometrics* 39:499–503; 1983.

### See Also

[spower](#), [ciapower](#), [bpower](#)

### Examples

```
#In this example, 4 plots are drawn on one page, one plot for each
#combination of noncompliance percentage. Within a plot, the
#5-year mortality % in the control group is on the x-axis, and
#separate curves are drawn for several % reductions in mortality
#with the intervention. The accrual period is 1.5y, with all
#patients followed at least 5y and some 6.5y.
```

```

par(mfrow=c(2,2),oma=c(3,0,3,0))

morts <- seq(10,25,length=50)
red <- c(10,15,20,25)

for(noncomp in c(0,10,15,-1)) {
  if(noncomp>=0) nc.i <- nc.c <- noncomp else {nc.i <- 25; nc.c <- 15}
  z <- paste("Drop-in ",nc.c,"%", Non-adherence ",nc.i,"%",sep="")
  plot(0,0,xlim=range(morts),ylim=c(0,1),
       xlab="5-year Mortality in Control Patients (%)",
       ylab="Power",type="n")
  title(z)
  cat(z,"\n")
  lty <- 0
  for(r in red) {
    lty <- lty+1
    power <- morts
    i <- 0
    for(m in morts) {
      i <- i+1
      power[i] <- cpower(5, 14000, m/100, r, 1.5, 5, nc.c, nc.i, pr=FALSE)
    }
    lines(morts, power, lty=lty)
  }
  if(noncomp==0)legend(18,.55,rev(paste(red,"% reduction",sep="")),
                     lty=4:1,bty="n")
}
mtitle("Power vs Non-Adherence for Main Comparison",
       ll="alpha=.05, 2-tailed, Total N=14000",cex.l=.8)
#
# Point sample size requirement vs. mortality reduction
# Root finder (uniroot()) assumes needed sample size is between
# 1000 and 40000
#
nc.i <- 25; nc.c <- 15; mort <- .18
red <- seq(10,25,by=.25)
samsiz <- red

i <- 0
for(r in red) {
  i <- i+1
  samsiz[i] <- uniroot(function(x) cpower(5, x, mort, r, 1.5, 5,
                                         nc.c, nc.i, pr=FALSE) - .8,
                      c(1000,40000))$root
}

samsiz <- samsiz/1000

```

```
par(mfrow=c(1,1))
plot(red, samsiz, xlab='% Reduction in 5-Year Mortality',
      ylab='Total Sample Size (Thousands)', type='n')
lines(red, samsiz, lwd=2)
title('Sample Size for Power=0.80\nDrop-in 15%, Non-adherence 25%')
title(sub='alpha=0.05, 2-tailed', adj=0)
```

---

Cs

*Character strings from unquoted names*

---

## Description

Cs makes a vector of character strings from a list of valid R names. .q is similar but also makes uses of names of arguments.

## Usage

```
Cs(...)  
.q(...)
```

## Arguments

... any number of names separated by commas. For .q any names of arguments will be used.

## Value

character string vector. For .q there will be a names attribute to the vector if any names appeared in ...

## See Also

sys.frame, deparse

## Examples

```
Cs(a,cat,dog)  
# subset.data.frame <- dataframe[,Cs(age,sex,race,bloodpressure,height)]  
.q(a, b, c, 'this and that')  
.q(dog=a, giraffe=b, cat=c)
```

---

 csv.get

 Read Comma-Separated Text Data Files
 

---

## Description

Read comma-separated text data files, allowing optional translation to lower case for variable names after making them valid S names. There is a facility for reading long variable labels as one of the rows. If labels are not specified and a final variable name is not the same as that in the header, the original variable name is saved as a variable label. Uses `read.csv` if the `data.table` package is not in effect, otherwise calls `fread`.

## Usage

```
csv.get(file, lowernames=FALSE, datevars=NULL, datetimevars=NULL,
        dateformat='%F',
        fixdates=c('none','year'), comment.char="", autodate=TRUE,
        allow=NULL, charfactor=FALSE,
        sep=',', skip=0, vnames=NULL, labels=NULL, text=NULL, ...)
```

## Arguments

<code>file</code>	the file name for import.
<code>lowernames</code>	set this to <code>TRUE</code> to change variable names to lower case.
<code>datevars</code>	character vector of names (after <code>lowernames</code> is applied) of variables to consider as a factor or character vector containing dates in a format matching <code>dateformat</code> . The default is <code>"%F"</code> which uses the <code>yyyy-mm-dd</code> format.
<code>datetimevars</code>	character vector of names (after <code>lowernames</code> is applied) of variables to consider to be date-time variables, with date formats as described under <code>datevars</code> followed by a space followed by time in <code>hh:mm:ss</code> format. <code>chron</code> is used to store such variables. If all times in the variable are <code>00:00:00</code> the variable will be converted to an ordinary date variable.
<code>dateformat</code>	for <code>cleanup.import</code> is the input format (see <a href="#">strptime</a> )
<code>fixdates</code>	for any of the variables listed in <code>datevars</code> that have a <code>dateformat</code> that <code>cleanup.import</code> understands, specifying <code>fixdates</code> allows corrections of certain formatting inconsistencies before the fields are attempted to be converted to dates (the default is to assume that the <code>dateformat</code> is followed for all observation for <code>datevars</code> ). Currently <code>fixdates='year'</code> is implemented, which will cause 2-digit or 4-digit years to be shifted to the alternate number of digits when <code>dateform</code> is the default <code>"%F"</code> or is <code>"%y-%m-%d"</code> , <code>"%m/%d/%y"</code> , or <code>"%m/%d/%Y"</code> . Two-digits years are padded with <code>20</code> on the left. Set <code>dateformat</code> to the desired format, not the exceptional format.
<code>comment.char</code>	a character vector of length one containing a single character or an empty string. Use <code>""</code> to turn off the interpretation of comments altogether.
<code>autodate</code>	Set to true to allow function to guess at which variables are dates



allow	a vector of characters allowed by R that should not be converted to periods in variable names. By default, underscores in variable names are converted to periods as with R before version 1.9.
charfactor	set to TRUE to change character variables to factors if they have fewer than n/2 unique values. Blanks and null strings are converted to NAs.
sep	field separator, defaults to comma
skip	number of records to skip before data start. Required if vnames or labels is given.
vnames	number of row containing variable names, default is one
labels	number of row containing variable labels, default is no labels
text	a character string containing the .csv file to use instead of file=. Passed to read.csv as the text= argument.
...	arguments to pass to read.csv other than skip and sep.

### Details

csv.get reads comma-separated text data files, allowing optional translation to lower case for variable names after making them valid S names. Original possibly non-legal names are taken to be variable labels if labels is not specified. Character or factor variables containing dates can be converted to date variables. cleanup.import is invoked to finish the job.

### Value

a new data frame.

### Author(s)

Frank Harrell, Vanderbilt University

### See Also

[sas.get](#), [data.frame](#), [cleanup.import](#), [read.csv](#), [strptime](#), [POSIXct](#), [Date](#), [fread](#)

### Examples

```
## Not run:
dat <- csv.get('myfile.csv')

# Read a csv file with junk in the first row, variable names in the
# second, long variable labels in the third, and junk in the 4th row
dat <- csv.get('myfile.csv', vnames=2, labels=3, skip=4)

## End(Not run)
```

**Description**

curveRep finds representative curves from a relatively large collection of curves. The curves usually represent time-response profiles as in serial (longitudinal or repeated) data with possibly unequal time points and greatly varying sample sizes per subject. After excluding records containing missing x or y, records are first stratified into kn groups having similar sample sizes per curve (subject). Within these strata, curves are next stratified according to the distribution of x points per curve (typically measurement times per subject). The [clara](#) clustering/partitioning function is used to do this, clustering on one, two, or three x characteristics depending on the minimum sample size in the current interval of sample size. If the interval has a minimum number of unique values of one, clustering is done on the single x values. If the minimum number of unique x values is two, clustering is done to create groups that are similar on both  $\min(x)$  and  $\max(x)$ . For groups containing no fewer than three unique x values, clustering is done on the trio of values  $\min(x)$ ,  $\max(x)$ , and the longest gap between any successive x. Then within sample size and x distribution strata, clustering of time-response profiles is based on p values of y all evaluated at the same p equally-spaced x's within the stratum. An option allows per-curve data to be smoothed with [lowess](#) before proceeding. Outer x values are taken as extremes of x across all curves within the stratum. Linear interpolation within curves is used to estimate y at the grid of x's. For curves within the stratum that do not extend to the most extreme x values in that stratum, extrapolation uses flat lines from the observed extremes in the curve unless `extrap=TRUE`. The p y values are clustered using [clara](#).

`print` and `plot` methods show results. By specifying an auxiliary `idcol` variable to `plot`, other variables such as treatment may be depicted to allow the analyst to determine for example whether subjects on different treatments are assigned to different time-response profiles. To write the frequencies of a variable such as treatment in the upper left corner of each panel (instead of the grand total number of clusters in that panel), specify `freq`.

`curveSmooth` takes a set of curves and smooths them using [lowess](#). If the number of unique x points in a curve is less than p, the smooth is evaluated at the unique x values. Otherwise it is evaluated at an equally spaced set of x points over the observed range. If fewer than 3 unique x values are in a curve, those points are used and smoothing is not done.

**Usage**

```
curveRep(x, y, id, kn = 5, kxdist = 5, k = 5, p = 5,
         force1 = TRUE, metric = c("euclidean", "manhattan"),
         smooth=FALSE, extrap=FALSE, pr=FALSE)

## S3 method for class 'curveRep'
print(x, ...)

## S3 method for class 'curveRep'
plot(x, which=1:length(res),
     method=c('all','lattice','data'),
```

```

m=NULL, probs=c(.5, .25, .75), nx=NULL, fill=TRUE,
idcol=NULL, freq=NULL, plotfreq=FALSE,
xlim=range(x), ylim=range(y),
xlab='x', ylab='y', colorfreq=FALSE, ...)
curveSmooth(x, y, id, p=NULL, pr=TRUE)

```

## Arguments

x	a numeric vector, typically measurement times. For <code>plot.curveRep</code> is an object created by <code>curveRep</code> .
y	a numeric vector of response values
id	a vector of curve (subject) identifiers, the same length as x and y
kn	number of curve sample size groups to construct. <code>curveRep</code> tries to divide the data into equal numbers of curves across sample size intervals.
kxdist	maximum number of x-distribution clusters to derive using <code>clara</code>
k	maximum number of x-y profile clusters to derive using <code>clara</code>
p	number of x points at which to interpolate y for profile clustering. For <code>curveSmooth</code> is the number of equally spaced points at which to evaluate the <code>lowess</code> smooth, and if p is omitted the smooth is evaluated at the original x values (which will allow <code>curveRep</code> to still know the x distribution)
force1	By default if any curves have only one point, all curves consisting of one point will be placed in a separate stratum. To prevent this separation, set <code>force1 = FALSE</code> .
metric	see <code>clara</code>
smooth	By default, linear interpolation is used on raw data to obtain y values to cluster to determine x-y profiles. Specify <code>smooth = TRUE</code> to replace observed points with <code>lowess</code> before computing y points on the grid. Also, when <code>smooth</code> is used, it may be desirable to use <code>extrap=TRUE</code> .
extrap	set to <code>TRUE</code> to use linear extrapolation to evaluate y points for x-y clustering. Not recommended unless smoothing has been or is being done.
pr	set to <code>TRUE</code> to print progress notes
which	an integer vector specifying which sample size intervals to plot. Must be specified if <code>method='lattice'</code> and must be a single number in that case.
method	The default makes individual plots of possibly all x-distribution by sample size by cluster combinations. Fewer may be plotted by specifying <code>which</code> . Specify <code>method='lattice'</code> to show a lattice xyplot of a single sample size interval, with x distributions going across and clusters going down. To not plot but instead return a data frame for a single sample size interval, specify <code>method='data'</code>
m	the number of curves in a cluster to randomly sample if there are more than m in a cluster. Default is to draw all curves in a cluster. For <code>method = "lattice"</code> you can specify <code>m = "quantiles"</code> to use the <code>xyplot</code> function to show quantiles of y as a function of x, with the quantiles specified by the <code>probs</code> argument. This cannot be used to draw a group containing <code>n = 1</code> .

<code>nx</code>	applies if <code>m = "quantiles"</code> . See <code>xYplot</code> .
<code>probs</code>	3-vector of probabilities with the central quantile first. Default uses quartiles.
<code>fill</code>	for <code>method = "all"</code> , by default if a sample size x-distribution stratum did not have enough curves to stratify into <code>k</code> x-y profiles, empty graphs are drawn so that a matrix of graphs will have the next row starting with a different sample size range or x-distribution. See the example below.
<code>idcol</code>	a named vector to be used as a table lookup for color assignments (does not apply when <code>m = "quantile"</code> ). The names of this vector are curve ids and the values are color names or numbers.
<code>freq</code>	a named vector to be used as a table lookup for a grouping variable such as treatment. The names are curve ids and values are any values useful for grouping in a frequency tabulation.
<code>plotfreq</code>	set to <code>TRUE</code> to plot the frequencies from the <code>freq</code> variable as horizontal bars instead of printing them. Applies only to <code>method = "lattice"</code> . By default the largest bar is 0.1 times the length of a panel's x-axis. Specify <code>plotfreq = 0.5</code> for example to make the longest bar half this long.
<code>colorfreq</code>	set to <code>TRUE</code> to color the frequencies printed by <code>plotfreq</code> using the colors provided by <code>idcol</code> .
<code>xlim, ylim, xlab, ylab</code>	plotting parameters. Default ranges are the ranges in the entire set of raw data given to <code>curveRep</code> .
<code>...</code>	arguments passed to other functions.

### Details

In the graph titles for the default graphic output, `n` refers to the minimum sample size, `x` refers to the sequential x-distribution cluster, and `c` refers to the sequential x-y profile cluster. Graphs from `method = "lattice"` are produced by `xYplot` and in the panel titles `distribution` refers to the x-distribution stratum and `cluster` refers to the x-y profile cluster.

### Value

a list of class "curveRep" with the following elements

<code>res</code>	a hierarchical list first split by sample size intervals, then by x distribution clusters, then containing a vector of cluster numbers with id values as a names attribute
<code>ns</code>	a table of frequencies of sample sizes per curve after removing NAs
<code>nomit</code>	total number of records excluded due to NAs
<code>missfreq</code>	a table of frequencies of number of NAs excluded per curve
<code>ncuts</code>	cut points for sample size intervals
<code>kn</code>	number of sample size intervals
<code>kxdist</code>	number of clusters on x distribution
<code>k</code>	number of clusters of curves within sample size and distribution groups
<code>p</code>	number of points at which to evaluate each curve for clustering

x  
y  
id                   input data after removing NAs

curveSmooth returns a list with elements x, y, id.

### Note

The references describe other methods for deriving representative curves, but those methods were not used here. The last reference which used a cluster analysis on principal components motivated curveRep however. The kml package does k-means clustering of longitudinal data with imputation.

### Author(s)

Frank Harrell  
Department of Biostatistics  
Vanderbilt University  
<fh@fharrell.com>

### References

Segal M. (1994): Representative curves for longitudinal data via regression trees. *J Comp Graph Stat* 3:214-233.

Jones MC, Rice JA (1992): Displaying the important features of large collections of similar curves. *Am Statistician* 46:140-145.

Zheng X, Simpson JA, et al (2005): Data from a study of effectiveness suggested potential prognostic factors related to the patterns of shoulder pain. *J Clin Epi* 58:823-830.

### See Also

[clara.dataRep](#)

### Examples

```
## Not run:
# Simulate 200 curves with per-curve sample sizes ranging from 1 to 10
# Make curves with odd-numbered IDs have an x-distribution that is random
# uniform [0,1] and those with even-numbered IDs have an x-dist. that is
# half as wide but still centered at 0.5. Shift y values higher with
# increasing IDs
set.seed(1)
N <- 200
nc <- sample(1:10, N, TRUE)
id <- rep(1:N, nc)
x <- y <- id
for(i in 1:N) {
  x[id==i] <- if(i %% 2) runif(nc[i]) else runif(nc[i], c(.25, .75))
  y[id==i] <- i + 10*(x[id==i] - .5) + runif(nc[i], -10, 10)
}
```

```

w <- curveRep(x, y, id, kxdist=2, p=10)
w
par(ask=TRUE, mfrow=c(4,5))
plot(w) # show everything, profiles going across
par(mfrow=c(2,5))
plot(w,1) # show n=1 results
# Use a color assignment table, assigning low curves to green and
# high to red. Unique curve (subject) IDs are the names of the vector.
cols <- c(rep('green', N/2), rep('red', N/2))
names(cols) <- as.character(1:N)
plot(w, 3, idcol=cols)
par(ask=FALSE, mfrow=c(1,1))

plot(w, 1, 'lattice') # show n=1 results
plot(w, 3, 'lattice') # show n=4-5 results
plot(w, 3, 'lattice', idcol=cols) # same but different color mapping
plot(w, 3, 'lattice', m=1) # show a single "representative" curve
# Show median, 10th, and 90th percentiles of supposedly representative curves
plot(w, 3, 'lattice', m='quantiles', probs=c(.5,.1,.9))
# Same plot but with much less grouping of x variable
plot(w, 3, 'lattice', m='quantiles', probs=c(.5,.1,.9), nx=2)

# Use ggplot2 for one sample size interval
z <- plot(w, 2, 'data')
require(ggplot2)
ggplot(z, aes(x, y, color=curve)) + geom_line() +
  facet_grid(distribution ~ cluster) +
  theme(legend.position='none') +
  labs(caption=z$ninterval[1])

# Smooth data before profiling. This allows later plotting to plot
# smoothed representative curves rather than raw curves (which
# specifying smooth=TRUE to curveRep would do, if curveSmooth was not used)
d <- curveSmooth(x, y, id)
w <- with(d, curveRep(x, y, id))

# Example to show that curveRep can cluster profiles correctly when
# there is no noise. In the data there are four profiles - flat, flat
# at a higher mean y, linearly increasing then flat, and flat at the
# first height except for a sharp triangular peak

set.seed(1)
x <- 0:100
m <- length(x)
profile <- matrix(NA, nrow=m, ncol=4)
profile[,1] <- rep(0, m)
profile[,2] <- rep(3, m)
profile[,3] <- c(0:3, rep(3, m-4))
profile[,4] <- c(0,1,3,1,rep(0,m-4))
col <- c('black','blue','green','red')
matplot(x, profile, type='l', col=col)
xeval <- seq(0, 100, length.out=5)

```

```

s <- x
matplot(x[s], profile[s,], type='l', col=col)

id <- rep(1:100, each=m)
X <- Y <- id
cols <- character(100)
names(cols) <- as.character(1:100)
for(i in 1:100) {
  s <- id==i
  X[s] <- x
  j <- sample(1:4,1)
  Y[s] <- profile[,j]
  cols[i] <- col[j]
}
table(cols)
yl <- c(-1,4)
w <- curveRep(X, Y, id, kn=1, kxdist=1, k=4)
plot(w, 1, 'lattice', idcol=cols, ylim=yl)
# Found 4 clusters but two have same profile
w <- curveRep(X, Y, id, kn=1, kxdist=1, k=3)
plot(w, 1, 'lattice', idcol=cols, freq=cols, plotfreq=TRUE, ylim=yl)
# Incorrectly combined black and red because default value p=5 did
# not result in different profiles at x=xeval
w <- curveRep(X, Y, id, kn=1, kxdist=1, k=4, p=40)
plot(w, 1, 'lattice', idcol=cols, ylim=yl)
# Found correct clusters because evaluated curves at 40 equally
# spaced points and could find the sharp triangular peak in profile 4

## End(Not run)

```

---

cut2

*Cut a Numeric Variable into Intervals*


---

### Description

cut2 is a function like cut but left endpoints are inclusive and labels are of the form [lower, upper), except that last interval is [lower, upper]. If cuts are given, will by default make sure that cuts include entire range of x. Also, if cuts are not given, will cut x into quantile groups (g given) or groups with a given minimum number of observations (m). Whereas cut creates a category object, cut2 creates a factor object. m is not guaranteed but is a target.

cutGn guarantees that the grouped variable will have a minimum of m observations in any group. This is done by an exhaustive algorithm that runs fast due to being coded in Fortran.

### Usage

```
cut2(x, cuts, m=150, g, levels.mean=FALSE, digits, minmax=TRUE,
oneval=TRUE, onlycuts=FALSE, formatfun=format, ...)
```

```
cutGn(x, m, what=c('mean', 'factor', 'summary', 'cuts', 'function'), rcode=FALSE)
```

**Arguments**

<code>x</code>	numeric vector to classify into intervals
<code>cuts</code>	cut points
<code>m</code>	desired minimum number of observations in a group. The algorithm does not guarantee that all groups will have at least <code>m</code> observations.
<code>g</code>	number of quantile groups
<code>levels.mean</code>	set to TRUE to make the new categorical vector have levels attribute that is the group means of <code>x</code> instead of interval endpoint labels
<code>digits</code>	number of significant digits to use in constructing levels. Default is 3 (5 if <code>levels.mean=TRUE</code> )
<code>minmax</code>	if <code>cuts</code> is specified but <code>min(x)&lt;min(cuts)</code> or <code>max(x)&gt;max(cuts)</code> , augments <code>cuts</code> to include min and max <code>x</code>
<code>oneval</code>	if an interval contains only one unique value, the interval will be labeled with the formatted version of that value instead of the interval endpoints, unless <code>oneval=FALSE</code>
<code>onlycuts</code>	set to TRUE to only return the vector of computed cuts. This consists of the interior values plus outer ranges.
<code>formatfun</code>	formatting function, supports formula notation (if <code>rlang</code> is installed)
<code>...</code>	additional arguments passed to <code>formatfun</code>
<code>what</code>	specifies the kind of vector values to return from <code>cutGn</code> , the default being like <code>'levels.mean'</code> of <code>cut2</code> . Specify <code>'summary'</code> to return a numeric 3-column matrix that summarizes the intervals satisfying the <code>m</code> requirement. Use <code>what='cuts'</code> to only return the vector of computed cutpoints. To create a function that will recode the variable in play using the same intervals as computed by <code>cutGn</code> , specify <code>what='function'</code> . This function will have a <code>what</code> argument to allow the user to decide later whether to recode into interval means or into a factor variable.
<code>rcode</code>	set to TRUE to run the <code>cutgn</code> algorithm in R. This is useful for speed comparisons with the default compiled code.

**Value**

a factor variable with levels of the form `[a, b)` or formatted means (character strings) unless `onlycuts` is TRUE in which case a numeric vector is returned

**See Also**

[cut](#), [quantile](#), [combine.levels](#)

**Examples**

```
set.seed(1)
x <- runif(1000, 0, 100)
z <- cut2(x, c(10,20,30))
table(z)
table(cut2(x, g=10))      # quantile groups
```



```

table(cut2(x, m=50))      # group x into intervals with at least 50 obs.

table(cutGn(x, m=50, what='factor'))
f <- cutGn(x, m=50, what='function')
f
f(c(-1, 2, 10), what='mean')
f(c(-1, 2, 10), what='factor')
## Not run:
x <- round(runif(200000), 3)
system.time(a <- cutGn(x, m=20))      # 0.02s
system.time(b <- cutGn(x, m=20, rcode=TRUE)) # 1.51s
identical(a, b)

## End(Not run)

```

---

data.frame.create.modify.check

*Tips for Creating, Modifying, and Checking Data Frames*

---

## Description

This help file contains a template for importing data to create an R data frame, correcting some problems resulting from the import and making the data frame be stored more efficiently, modifying the data frame (including better annotating it and changing the names of some of its variables), and checking and inspecting the data frame for reasonableness of the values of its variables and to describe patterns of missing data. Various built-in functions and functions in the Hmisc library are used. At the end some methods for creating data frames “from scratch” within R are presented.

The examples below attempt to clarify the separation of operations that are done on a data frame as a whole, operations that are done on a small subset of its variables without attaching the whole data frame, and operations that are done on many variables after attaching the data frame in search position one. It also tries to clarify that for analyzing several separate variables using R commands that do not support a data argument, it is helpful to attach the data frame in a search position later than position one.

It is often useful to create, modify, and process datasets in the following order.

1. Import external data into a data frame (if the raw data do not contain column names, provide these during the import if possible)
2. Make global changes to a data frame (e.g., changing variable names)
3. Change attributes or values of variables within a data frame
4. Do analyses involving the whole data frame (without attaching it)  
(Data frame still in .Data)
5. Do analyses of individual variables (after attaching the data frame in search position two or later)

## Details

The examples below use the FEV dataset from *Rosner 1995*. Almost any dataset would do. The jactable data are taken from *Galobardes, et al.*

Presently, giving a variable the "units" attribute (using the **Hmisc** `units` function) only benefits the **Hmisc** `describe` function and the **rms** library's version of the `link[rms]{Surv}` function. Variables labels defined with the **Hmisc** `label` function are used by `describe`, `summary.formula`, and many of the plotting functions in **Hmisc** and **rms**.

## References

Alzola CF, Harrell FE (2006): *An Introduction to S and the Hmisc and Design Libraries*. Chapters 3 and 4, <https://hbiostat.org/R/doc/sintro.pdf>.

Galobardes, et al. (1998), *J Clin Epi* 51:875-881.

Rosner B (1995): *Fundamentals of Biostatistics, 4th Edition*. New York: Duxbury Press.

## See Also

`scan`, `read.table`, `cleanup.import`, `sas.get`, `data.frame`, `attach`, `detach`, `describe`, `datadensity`, `plot.data.frame`, `hist.data.frame`, `naclus`, `factor`, `label`, `units`, `names`, `expand.grid`, `summary.formula`, `summary.data.frame`, `casefold`, `edit`, `page`, `plot.data.frame`, `Cs`, `combine.levels`, `upData`

## Examples

```
## Not run:
# First, we do steps that create or manipulate the data
# frame in its entirety. For S-Plus, these are done with
# .Data in search position one (the default at the
# start of the session).
#
# -----
# Step 1: Create initial draft of data frame
#
# We usually begin by importing a dataset from
# # another application. ASCII files may be imported
# using the scan and read.table functions. SAS
# datasets may be imported using the Hmisc sas.get
# function (which will carry more attributes from
# SAS than using File \dots Import) from the GUI
# menus. But for most applications (especially
# Excel), File \dots Import will suffice. If using
# the GUI, it is often best to provide variable
# names during the import process, using the Options
# tab, rather than renaming all fields later Of
# course, if the data to be imported already have
# field names (e.g., in Excel), let S use those
# automatically. If using S-Plus, you can use a
# command to execute File \dots Import, e.g.:

import.data(fileName = "/windows/temp/fev.asc",
```

```
        FileType = "ASCII", DataFrame = "FEV")

# Here we name the new data frame FEV rather than
# fev, because we wanted to distinguish a variable
# in the data frame named fev from the data frame
# name. For S-Plus the command will look
# instead like the following:

FEV <- importData("/tmp/fev.asc")

# -----
# Step 2: Clean up data frame / make it be more
# efficiently stored
#
# Unless using sas.get to import your dataset
# (sas.get already stores data efficiently), it is
# usually a good idea to run the data frame through
# the Hmisc cleanup.import function to change
# numeric variables that are always whole numbers to
# be stored as integers, the remaining numerics to
# single precision, strange values from Excel to
# NAs, and character variables that always contain
# legal numeric values to numeric variables.
# cleanup.import typically halves the size of the
# data frame. If you do not specify any parameters
# to cleanup.import, the function assumes that no
# numeric variable needs more than 7 significant
# digits of precision, so all non-integer-valued
# variables will be converted to single precision.

FEV <- cleanup.import(FEV)

# -----
# Step 3: Make global changes to the data frame
#
# A data frame has attributes that are "external" to
# its variables. There are the vector of its
# variable names ("names" attribute), the
# observation identifiers ("row.names"), and the
# "class" (whose value is "data.frame"). The
# "names" attribute is the one most commonly in need
# of modification. If we had wanted to change all
# the variable names to lower case, we could have
# specified lowernames=TRUE to the cleanup.import
```

```
# invocation above, or type

names(FEV) <- casefold(names(FEV))

# The upData function can also be used to change
# variable names in two ways (see below).
# To change names in a non-systematic way we use
# other options. Under Windows/NT the most
# straightforward approach is to change the names
# interactively. Click on the data frame in the
# left panel of the Object Browser, then in the
# right pane click twice (slowly) on a variable.
# Use the left arrow and other keys to edit the
# name. Click outside that name field to commit the
# change. You can also rename columns while in a
# Data Sheet. To instead use programming commands
# to change names, use something like:

names(FEV)[6] <- 'smoke' # assumes you know the positions!
names(FEV)[names(FEV)=='smoking'] <- 'smoke'
names(FEV) <- edit(names(FEV))

# The last example is useful if you are changing
# many names. But none of the interactive
# approaches such as edit() are handy if you will be
# re-importing the dataset after it is updated in
# its original application. This problem can be
# addressed by saving the new names in a permanent
# vector in .Data:

new.names <- names(FEV)

# Then if the data are re-imported, you can type

names(FEV) <- new.names

# to rename the variables.

# -----
# Step 4: Delete unneeded variables
#
# To delete some of the variables, you can
```

```

# right-click on variable names in the Object
# Browser's right pane, then select Delete. You can
# also set variables to have NULL values, which
# causes the system to delete them. We don't need
# to delete any variables from FEV but suppose we
# did need to delete some from mydframe.

mydframe$x1 <- NULL
mydframe$x2 <- NULL
mydframe[c('age','sex')] <- NULL # delete 2 variables
mydframe[Cs(age,sex)] <- NULL # same thing

# The last example uses the Hmisc short-cut quoting
# function Cs. See also the drop parameter to upData.

# -----
# Step 5: Make changes to individual variables
#       within the data frame
#
# After importing data, the resulting variables are
# seldom self - documenting, so we commonly need to
# change or enhance attributes of individual
# variables within the data frame.
#
# If you are only changing a few variables, it is
# efficient to change them directly without
# attaching the entire data frame.

FEV$sex <- factor(FEV$sex, 0:1, c('female','male'))
FEV$smoke <- factor(FEV$smoke, 0:1,
                   c('non-current smoker','current smoker'))
units(FEV$age) <- 'years'
units(FEV$fev) <- 'L'
label(FEV$fev) <- 'Forced Expiratory Volume'
units(FEV$height) <- 'inches'

# When changing more than one or two variables it is
# more convenient change the data frame using the
# Hmisc upData function.

FEV2 <- upData(FEV,
               rename=c(smoking='smoke'),
               # omit if renamed above
               drop=c('var1','var2'),
               levels=list(sex =list(female=0,male=1),

```

```

        smoke=list('non-current smoker'=0,
                   'current smoker'=1)),
        units=list(age='years', fev='L', height='inches'),
        labels=list(fev='Forced Expiratory Volume'))

# An alternative to levels=list(\dots) is for example
# upData(FEV, sex=factor(sex,0:1,c('female','male'))).
#
# Note that we saved the changed data frame into a
# new data frame FEV2. If we were confident of the
# correctness of our changes we could have stored
# the new data frame on top of the old one, under
# the original name FEV.

# -----
# Step 6: Check the data frame
#
# The Hmisc describe function is perhaps the first
# function that should be used on the new data
# frame. It provides documentation of all the
# variables and the frequency tabulation, counts of
# NAs, and 5 largest and smallest values are
# helpful in detecting data errors. Typing
# describe(FEV) will write the results to the
# current output window. To put the results in a
# new window that can persist, even upon exiting
# S, we use the page function. The describe
# output can be minimized to an icon but kept ready
# for guiding later steps of the analysis.

page(describe(FEV2), multi=TRUE)
# multi=TRUE allows that window to persist while
# control is returned to other windows

# The new data frame is OK. Store it on top of the
# old FEV and then use the graphical user interface
# to delete FEV2 (click on it and hit the Delete
# key) or type rm(FEV2) after the next statement.

FEV <- FEV2

# Next, we can use a variety of other functions to
# check and describe all of the variables. As we
# are analyzing all or almost all of the variables,
# this is best done without attaching the data
# frame. Note that plot.data.frame plots inverted
# CDFs for continuous variables and dot plots

```

```
# showing frequency distributions of categorical
# ones.

summary(FEV)
# basic summary function (summary.data.frame)

plot(FEV)          # plot.data.frame
datadensity(FEV)
# rug plots and freq. bar charts for all var.

hist.data.frame(FEV)
# for variables having > 2 values

by(FEV, FEV$smoke, summary)
# use basic summary function with stratification

# -----
# Step 7: Do detailed analyses involving individual
#         variables
#
# Analyses based on the formula language can use
# data= so attaching the data frame may not be
# required. This saves memory. Here we use the
# Hmisc summary.formula function to compute 5
# statistics on height, stratified separately by age
# quartile and by sex.

options(width=80)
summary(height ~ age + sex, data=FEV,
        fun=function(y)c(smean.sd(y),
                        smedian.hilow(y,conf.int=.5)))
# This computes mean height, S.D., median, outer quartiles

fit <- lm(height ~ age*sex, data=FEV)
summary(fit)

# For this analysis we could also have attached the
# data frame in search position 2. For other
# analyses, it is mandatory to attach the data frame
# unless FEV$ prefixes each variable name.
# Important: DO NOT USE attach(FEV, 1) or
# attach(FEV, pos=1, \dots) if you are only analyzing
# and not changing the variables, unless you really
```

```
# need to avoid conflicts with variables in search
# position 1 that have the same names as the
# variables in FEV. Attaching into search position
# 1 will cause S-Plus to be more of a memory hog.

attach(FEV)
# Use e.g. attach(FEV[,Cs(age,sex)]) if you only
# want to analyze a small subset of the variables
# Use e.g. attach(FEV[FEV$sex=='male',]) to
# analyze a subset of the observations

summary(height ~ age + sex,
         fun=function(y)c(smean.sd(y),
                        smedian.hilow(y,conf.int=.5)))
fit <- lm(height ~ age*sex)

# Run generic summary function on height and fev,
# stratified by sex
by(data.frame(height,fev), sex, summary)

# Cross-classify into 4 sex x smoke groups
by(FEV, list(sex,smoke), summary)

# Plot 5 quantiles
s <- summary(fev ~ age + sex + height,
             fun=function(y)quantile(y,c(.1,.25,.5,.75,.9)))

plot(s, which=1:5, pch=c(1,2,15,2,1), #pch=c('=', '[' , 'o', ']' , '='),
     main='A Discovery', xlab='FEV')

# Use the nonparametric bootstrap to compute a
# 0.95 confidence interval for the population mean fev
smean.cl.boot(fev) # in Hmisc

# Use the Statistics \dots Compare Samples \dots One Sample
# keys to get a normal-theory-based C.I. Then do it
# more manually. The following method assumes that
# there are no NAs in fev

sd <- sqrt(var(fev))
xbar <- mean(fev)
xbar
sd
n <- length(fev)
```



```
qt(.975,n-1)
# prints 0.975 critical value of t dist. with n-1 d.f.
```

```
xbar + c(-1,1)*sd/sqrt(n)*qt(.975,n-1)
# prints confidence limits
```

```
# Fit a linear model
# fit <- lm(fev ~ other variables \dots)
```

```
detach()
```

```
# The last command is only needed if you want to
# start operating on another data frame and you want
# to get FEV out of the way.
```

```
# -----
# Creating data frames from scratch
#
# Data frames can be created from within S. To
# create a small data frame containing ordinary
# data, you can use something like
```

```
dframe <- data.frame(age=c(10,20,30),
                     sex=c('male','female','male'),
                     stringsAsFactors=TRUE)
```

```
# You can also create a data frame using the Data
# Sheet. Create an empty data frame with the
# correct variable names and types, then edit in the
# data.
```

```
dd <- data.frame(age=numeric(0),sex=character(0),
                 stringsAsFactors=TRUE)
```

```
# The sex variable will be stored as a factor, and
# levels will be automatically added to it as you
# define new values for sex in the Data Sheet's sex
# column.
#
# When the data frame you need to create is defined
# by systematically varying variables (e.g., all
# possible combinations of values of each variable),
```

```
# the expand.grid function is useful for quickly
# creating the data. Then you can add
# non-systematically-varying variables to the object
# created by expand.grid, using programming
# statements or editing the Data Sheet. This
# process is useful for creating a data frame
# representing all the values in a printed table.
# In what follows we create a data frame
# representing the combinations of values from an 8
# x 2 x 2 x 2 (event x method x sex x what) table,
# and add a non-systematic variable percent to the
# data.
```

```
jcetable <- expand.grid(
  event=c('Wheezing at any time',
          'Wheezing and breathless',
          'Wheezing without a cold',
          'Waking with tightness in the chest',
          'Waking with shortness of breath',
          'Waking with an attack of cough',
          'Attack of asthma',
          'Use of medication'),
  method=c('Mail', 'Telephone'),
  sex=c('Male', 'Female'),
  what=c('Sensitivity', 'Specificity'))
```

```
jcetable$percent <-
c(756,618,706,422,356,578,289,333,
  576,421,789,273,273,212,212,212,
  613,763,713,403,377,541,290,226,
  613,684,632,290,387,613,258,129,
  656,597,438,780,732,679,938,919,
  714,600,494,877,850,703,963,987,
  755,420,480,794,779,647,956,941,
  766,423,500,833,833,604,955,986) / 10
```

```
# In jcetable, event varies most rapidly, then
# method, then sex, and what.
```

```
## End(Not run)
```

---

dataRep

*Representativeness of Observations in a Data Set*

---

## Description

These functions are intended to be used to describe how well a given set of new observations (e.g., new subjects) were represented in a dataset used to develop a predictive model. The dataRep

function forms a data frame that contains all the unique combinations of variable values that existed in a given set of variable values. Cross-classifications of values are created using exact values of variables, so for continuous numeric variables it is often necessary to round them to the nearest  $v$  and to possibly curtail the values to some lower and upper limit before rounding. Here  $v$  denotes a numeric constant specifying the matching tolerance that will be used. `dataRep` also stores marginal distribution summaries for all the variables. For numeric variables, all 101 percentiles are stored, and for all variables, the frequency distributions are also stored (frequencies are computed after any rounding and curtailment of numeric variables). For the purposes of rounding and curtailment, the `roundN` function is provided. A `print` method will summarize the calculations made by `dataRep`, and if `long=TRUE` all unique combinations of values and their frequencies in the original dataset are printed.

The `predict` method for `dataRep` takes a new data frame having variables named the same as the original ones (but whose factor levels are not necessarily in the same order) and examines the collapsed cross-classifications created by `dataRep` to find how many observations were similar to each of the new observations after any rounding or curtailment of limits is done. `predict` also does some calculations to describe how the variable values of the new observations "stack up" against the marginal distributions of the original data. For categorical variables, the percent of observations having a given variable with the value of the new observation (after rounding for variables that were through `roundN` in the formula given to `dataRep`) is computed. For numeric variables, the percentile of the original distribution in which the current value falls will be computed. For this purpose, the data are not rounded because the 101 original percentiles were retained; linear interpolation is used to estimate percentiles for values between two tabulated percentiles. The lowest marginal frequency of matching values across all variables is also computed. For example, if an age, sex combination matches 10 subjects in the original dataset but the age value matches 100 ages (after rounding) and the sex value matches the sex code of 300 observations, the lowest marginal frequency is 100, which is a "best case" upper limit for multivariable matching. I.e., matching on all variables has to result on a lower frequency than this amount. A `print` method for the output of `predict.dataRep` prints all calculations done by `predict` by default. Calculations can be selectively suppressed.

### Usage

```
dataRep(formula, data, subset, na.action)

roundN(x, tol=1, clip=NULL)

## S3 method for class 'dataRep'
print(x, long=FALSE, ...)

## S3 method for class 'dataRep'
predict(object, newdata, ...)

## S3 method for class 'predict.dataRep'
print(x, prdata=TRUE, prpct=TRUE, ...)
```

### Arguments

`formula` a formula with no left-hand-side. Continuous numeric variables in need of rounding should appear in the formula as e.g. `roundN(x, 5)` to have a tolerance

	of e.g. +/- 2.5 in matching. Factor or character variables as well as numeric ones not passed through <code>roundN</code> are matched on exactly.
<code>x</code>	a numeric vector or an object created by <code>dataRep</code>
<code>object</code>	the object created by <code>dataRep</code> or <code>predict.dataRep</code>
<code>data, subset, na.action</code>	standard modeling arguments. Default <code>na.action</code> is <code>na.delete</code> , i.e., observations in the original dataset having any variables missing are deleted up front.
<code>tol</code>	rounding constant (tolerance is actually <code>tol/2</code> as values are rounded to the nearest <code>tol</code> )
<code>clip</code>	a 2-vector specifying a lower and upper limit to curtail values of <code>x</code> before rounding
<code>long</code>	set to <code>TRUE</code> to see all unique combinations and frequency count
<code>newdata</code>	a data frame containing all the variables given to <code>dataRep</code> but not necessarily in the same order or having factor levels in the same order
<code>prdata</code>	set to <code>FALSE</code> to suppress printing <code>newdata</code> and the count of matching observations (plus the worst-case marginal frequency).
<code>prpct</code>	set to <code>FALSE</code> to not print percentiles and percents
<code>...</code>	unused

### Value

`dataRep` returns a list of class "dataRep" containing the collapsed data frame and frequency counts along with marginal distribution information. `predict` returns an object of class "predict.dataRep" containing information determined by matching observations in `newdata` with the original (collapsed) data.

### Side Effects

`print.dataRep` prints.

### Author(s)

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University School of Medicine  
 <fh@fharrell.com>

### See Also

[round](#), [table](#)

### Examples

```
set.seed(13)
num.symptoms <- sample(1:4, 1000, TRUE)
sex <- factor(sample(c('female', 'male'), 1000, TRUE))
x <- runif(1000)
```

```
x[1] <- NA
table(num.symptoms, sex, .25*round(x/.25))

d <- dataRep(~ num.symptoms + sex + roundN(x,.25))
print(d, long=TRUE)

predict(d, data.frame(num.symptoms=1:3, sex=c('male','male','female'),
                    x=c(.03,.5,1.5)))
```

---

deff

*Design Effect and Intra-cluster Correlation*

---

### Description

Computes the Kish design effect and corresponding intra-cluster correlation for a single cluster-sampled variable

### Usage

```
deff(y, cluster)
```

### Arguments

y	variable to analyze
cluster	a variable whose unique values indicate cluster membership. Any type of variable is allowed.

### Value

a vector with named elements n (total number of non-missing observations), clusters (number of clusters after deleting missing data), rho(intra-cluster correlation), and deff (design effect).

### Author(s)

Frank Harrell  
Department of Biostatistics  
Vanderbilt University  
<fh@fharrell.com>

### See Also

[bootcov](#), [robcov](#)

## Examples

```
set.seed(1)
blood.pressure <- rnorm(1000, 120, 15)
clinic <- sample(letters, 1000, replace=TRUE)
deff(blood.pressure, clinic)
```

---

describe

*Concise Statistical Description of a Vector, Matrix, Data Frame, or Formula*

---

## Description

describe is a generic method that invokes `describe.data.frame`, `describe.matrix`, `describe.vector`, or `describe.formula`. `describe.vector` is the basic function for handling a single variable. This function determines whether the variable is character, factor, category, binary, discrete numeric, and continuous numeric, and prints a concise statistical summary according to each. A numeric variable is deemed discrete if it has  $\leq 10$  distinct values. In this case, quantiles are not printed. A frequency table is printed for any non-binary variable if it has no more than 20 distinct values. For any variable for which the frequency table is not printed, the 5 lowest and highest values are printed. This behavior can be overridden for long character variables with many levels using the `listunique` parameter, to get a complete tabulation.

describe is especially useful for describing data frames created by `*.get`, as labels, formats, value labels, and (in the case of `sas.get`) frequencies of special missing values are printed.

For a binary variable, the sum (number of 1's) and mean (proportion of 1's) are printed. If the first argument is a formula, a model frame is created and passed to `describe.data.frame`. If a variable is of class "impute", a count of the number of imputed values is printed. If a date variable has an attribute `partial.date` (this is set up by `sas.get`), counts of how many partial dates are actually present (missing month, missing day, missing both) are also presented. If a variable was created by the special-purpose function `substi` (which substitutes values of a second variable if the first variable is NA), the frequency table of substitutions is also printed.

For numeric variables, describe adds an item called `Info` which is a relative information measure using the relative efficiency of a proportional odds/Wilcoxon test on the variable relative to the same test on a variable that has no ties. `Info` is related to how continuous the variable is, and ties are less harmful the more untied values there are. The formula for `Info` is one minus the sum of the cubes of relative frequencies of values divided by one minus the square of the reciprocal of the sample size. The lowest information comes from a variable having only one distinct value following by a highly skewed binary variable. `Info` is reported to two decimal places.

A latex method exists for converting the describe object to a LaTeX file. For numeric variables having more than 20 distinct values, describe saves in its returned object the frequencies of 100 evenly spaced bins running from minimum observed value to the maximum. When there are less than or equal to 20 distinct values, the original values are maintained. `latex` and `html` insert a spike histogram displaying these frequency counts in the tabular material using the LaTeX `picture` environment. For example output see <https://hbiostat.org/doc/rms/book/chapter7edition1.pdf>. Note that the latex method assumes you have the following styles installed in your latex installation: `setspace` and `relsize`.

The `html` method mimics the LaTeX output. This is useful in the context of Quarto/Rmarkdown `html` and `html` notebook output. If `options(prType='html')` is in effect, calling `print` on an object that is the result of running `describe` on a data frame will result in rendering the HTML version. If run from the console a browser window will open. When `which` is specified to `print`, whether or not `prType='html'` is in effect, a `gt` package `html` table will be produced containing only the types of variables requested. When `which='both'` a list with element names `Continuous` and `Categorical` is produced, making it convenient for the user to print as desired, or to pass the list directed to the `qreport` `maketabs` function when using Quarto.

The `plot` method is for `describe` objects run on data frames. It produces spike histograms for a graphic of continuous variables and a dot chart for categorical variables, showing category proportions. The graphic format is `ggplot2` if the user has not set `options(grType='plotly')` or has set the `grType` option to something other than `'plotly'`. Otherwise `plotly` graphics that are interactive are produced, and these can be placed into an Rmarkdown `html` notebook. The user must install the `plotly` package for this to work. When the user hovers the mouse over a bin for a raw data value, the actual value will pop-up (formatted using digits). When the user hovers over the minimum data value, most of the information calculated by `describe` will pop up. For each variable, the number of missing values is used to assign the color to the histogram or dot chart, and a legend is drawn. Color is not used if there are no missing values in any variable. For categorical variables, hovering over the leftmost point for a variable displays details, and for all points proportions, numerators, and denominators are displayed in the popup. If both continuous and categorical variables are present and `which='both'` is specified, the `plot` method returns an unclassed list containing two objects, named `'Categorical'` and `'Continuous'`, in that order.

Sample weights may be specified to any of the functions, resulting in weighted means, quantiles, and frequency tables.

Note: As discussed in Cox and Longton (2008), Stata Technical Bulletin 8(4) pp. 557, the term "unique" has been replaced with "distinct" in the output (but not in parameter names).

When weights are not used, the pseudomedian and Gini's mean difference are computed for numeric variables. The pseudomedian is labeled `pMedian` and is the median of all possible pairwise averages. It is a robust and efficient measure of location that equals the mean and median for symmetric distributions. It is also called the Hodges-Lehmann one-sample estimator. Gini's mean difference is a robust measure of dispersion that is the mean absolute difference between any pairs of observations. In simple output Gini's difference is labeled `Gmd`.

`formatdescribeSingle` is a service function for `latex`, `html`, and `print` methods for single variables that is not intended to be called by the user.

## Usage

```
## S3 method for class 'vector'
describe(x, descript, exclude.missing=TRUE, digits=4,
         listunique=0, listnchar=12,
         weights=NULL, normwt=FALSE, minlength=NULL, shortmChoice=TRUE,
         rmhtml=FALSE, trans=NULL, lumptails=0.01, ...)
## S3 method for class 'matrix'
describe(x, descript, exclude.missing=TRUE, digits=4, ...)
## S3 method for class 'data.frame'
describe(x, descript, exclude.missing=TRUE,
         digits=4, trans=NULL, ...)
```

```

## S3 method for class 'formula'
describe(x, descript, data, subset, na.action,
         digits=4, weights, ...)
## S3 method for class 'describe'
print(x, which = c('both', 'categorical', 'continuous'), ...)
## S3 method for class 'describe'
latex(object, title=NULL,
       file=paste('describe',first.word(expr=attr(object,'descript')),'tex',sep='.'),
       append=FALSE, size='small', tabular=TRUE, greek=TRUE,
       spacing=0.7, lspace=c(0,0), ...)
## S3 method for class 'describe.single'
latex(object, title=NULL, vname,
       file, append=FALSE, size='small', tabular=TRUE, greek=TRUE,
       lspace=c(0,0), ...)
## S3 method for class 'describe'
html(object, size=85, tabular=TRUE,
     greek=TRUE, scroll=FALSE, rows=25, cols=100, ...)
## S3 method for class 'describe.single'
html(object, size=85,
     tabular=TRUE, greek=TRUE, ...)
formatdescribeSingle(x, condense=c('extremes', 'frequencies', 'both', 'none'),
                    lang=c('plain', 'latex', 'html'), verb=0, lspace=c(0, 0),
                    size=85, ...)
## S3 method for class 'describe'
plot(x, which=c('both', 'continuous', 'categorical'),
     what=NULL,
     sort=c('ascending', 'descending', 'none'),
     n.unique=10, digits=5, bvspace=2, ...)

```

## Arguments

<code>x</code>	a data frame, matrix, vector, or formula. For a data frame, the <code>describe.data.frame</code> function is automatically invoked. For a matrix, <code>describe.matrix</code> is called. For a formula, <code>describe.data.frame(model.frame(x))</code> is invoked. The formula may or may not have a response variable. For <code>print</code> , <code>latex</code> , <code>html</code> , or <code>formatdescribeSingle</code> , <code>x</code> is an object created by <code>describe</code> .
<code>descript</code>	optional title to print for <code>x</code> . The default is the name of the argument or the "label" attributes of individual variables. When the first argument is a formula, <code>descript</code> defaults to a character representation of the formula.
<code>exclude.missing</code>	set to <code>TRUE</code> to print the names of variables that contain only missing values. This list appears at the bottom of the printout, and no space is taken up for such variables in the main listing.
<code>digits</code>	number of significant digits to print. For <code>plot.describe</code> is the number of significant digits to put in hover text for <code>plotly</code> when showing raw variable values.
<code>listunique</code>	For a character variable that is not an <code>mChoice</code> variable, that has its longest string length greater than <code>listnchar</code> , and that has no more than <code>listunique</code> distinct values, all values are listed in alphabetic order. Any value having more than



one occurrence has the frequency of occurrence included. Specify `listunique` equal to some value at least as large as the number of observations to ensure that all character variables will have all their values listed. For purposes of tabulating character strings, multiple white spaces of any kind are translated to a single space, leading and trailing white space are ignored, and case is ignored.

<code>listnchar</code>	see <code>listunique</code>
<code>weights</code>	a numeric vector of frequencies or sample weights. Each observation will be treated as if it were sampled <code>weights</code> times.
<code>minlength</code>	value passed to <code>summary.mChoice</code>
<code>shortmChoice</code>	set to <code>FALSE</code> to have <code>summary</code> of <code>mChoice</code> variables use actual levels everywhere, instead of abbreviating to integers and printing of all original labels at the top
<code>rmhtml</code>	set to <code>TRUE</code> to strip html from variable labels
<code>trans</code>	for <code>describe.vector</code> is a list specifying how to transform <code>x</code> for constructing the frequency distribution used in spike histograms. The first element of the list is a character string describing the transformation, the second is the transformation function, and the third argument is the inverse of this function that is used in labeling points on the original scale, e.g. <code>trans=list('log', log, exp)</code> . For <code>describe.data.frame</code> <code>trans</code> is a list of such lists, with the name of each list being name of the variable to which the transformation applies. See <a href="https://hbiostat.org/rmsc/impred.html#data">https://hbiostat.org/rmsc/impred.html#data</a> for an example.
<code>lumptails</code>	specifies the quantile to use (its complement is also used) for grouping observations in the tails so that outliers have less chance of distorting the variable's range for sparkline spike histograms. The default is 0.01, i.e., observations below the 0.01 quantile are grouped together in the leftmost bin, and observations above the 0.99 quantile are grouped to form the last bin.
<code>normwt</code>	The default, <code>normwt=FALSE</code> results in the use of <code>weights</code> as weights in computing various statistics. In this case the sample size is assumed to be equal to the sum of <code>weights</code> . Specify <code>normwt=TRUE</code> to divide <code>weights</code> by a constant so that <code>weights</code> sum to the number of observations (length of vectors specified to <code>describe</code> ). In this case the number of observations is taken to be the actual number of records given to <code>describe</code> .
<code>object</code>	a result of <code>describe</code>
<code>title</code>	unused
<code>data</code>	a data frame, data table, or list
<code>subset</code>	a subsetting expression
<code>na.action</code>	These are used if a formula is specified. <code>na.action</code> defaults to <code>na.retain</code> which does not delete any NAs from the data frame. Use <code>na.action=na.omit</code> or <code>na.delete</code> to drop any observation with any NA before processing.
<code>...</code>	arguments passed to <code>describe.default</code> which are passed to calls to <code>format</code> for numeric variables. For example if using R <code>POSIXct</code> or <code>Date</code> date/time formats, specifying <code>describe(d, format='%d%b%y')</code> will print date/time variables as <code>"01Jan2000"</code> . This is useful for omitting the time component. See the help file for <code>format.POSIXct</code> or <code>format.Date</code> for more information. For plot methods, <code>...</code> is ignored. For <code>html</code> and <code>latex</code> methods, <code>...</code> is used to pass optional

arguments to `formatDescribeSingle`, especially the `condense` argument. For the `print` method when `which=` is given, possible arguments to use for tabulating continuous variable output are `sparkwidth` (the width of the spike histogram sparkline in pixels, defaulting to 200), `qcondense` (set to `FALSE` to devote separate columns to all quantiles), `extremes` (set to `TRUE` to print the 5 lowest and highest values in the table of continuous variables). For categorical variable output, the argument `freq` can be used to specify how frequency tables are rendered: `'chart'` (the default; an interactive sparkline frequency bar chart) or `freq='table'` for small tables. `sort` is another argument passed to `html_describe_cat`. For sparkline frequency charts the default is to sort non-numeric categories in descending order of frequency. Set `code=FALSE` to use the original data order. The `w` argument also applies to categorical variable output.

<code>file</code>	name of output file (should have a suffix of <code>.tex</code> ). Default name is formed from the first word of the <code>describe</code> element of the <code>describe</code> object, prefixed by <code>"describe"</code> . Set <code>file=""</code> to send LaTeX code to standard output instead of a file.
<code>append</code>	set to <code>TRUE</code> to have latex append text to an existing file named <code>file</code>
<code>size</code>	LaTeX text size ( <code>"small"</code> , the default, or <code>"normalsize"</code> , <code>"tiny"</code> , <code>"scriptsize"</code> , etc.) for the <code>describe</code> output in LaTeX. For <code>html</code> is the percent of the prevailing font size to use for the output.
<code>tabular</code>	set to <code>FALSE</code> to use <code>verbatim</code> rather than <code>tabular</code> (or <code>html table</code> ) environment for the summary statistics output. By default, <code>tabular</code> is used if the output is not too wide.
<code>greek</code>	By default, the <code>latex</code> and <code>html</code> methods will change names of greek letters that appear in variable labels to appropriate LaTeX symbols in math mode, or <code>html</code> symbols, unless <code>greek=FALSE</code> .
<code>spacing</code>	By default, the <code>latex</code> method for <code>describe</code> run on a matrix or data frame uses the <code>setspace</code> LaTeX package with a line spacing of 0.7 so as to no waste space. Specify <code>spacing=0</code> to suppress the use of the <code>setspace</code> 's <code>spacing</code> environment, or specify another positive value to use this environment with a different spacing.
<code>lspace</code>	extra vertical scape, in character size units (i.e., <code>"ex"</code> as appended to the space). When using certain font sizes, there is too much space left around LaTeX <code>verbatim</code> environments. This two-vector specifies space to remove (i.e., the values are negated in forming the <code>vspace</code> command) before (first element) and after (second element of <code>lspace</code> ) <code>verbatim</code> s
<code>scroll</code>	set to <code>TRUE</code> to create an <code>html</code> scrollable box for the <code>html</code> output
<code>rows, cols</code>	the number of rows or columns to allocate for the scrollable box
<code>vname</code>	unused argument in <code>latex.describe.single</code>
<code>which</code>	specifies whether to plot numeric continuous or binary/categorical variables, or both. When <code>"both"</code> a list with two elements is created. Each element is a <code>ggplot2</code> or <code>plotly</code> object. If there are no variables of a given type, a single <code>ggplot2</code> or <code>plotly</code> object is returned, ready to print. For <code>print.describe</code> may be <code>"categorical"</code> or <code>"continuous"</code> , causing a <code>gt</code> table to be created with the categorical or continuous variable <code>describe</code> results.
<code>what</code>	character or numeric vector specifying which variables to plot; default is to plot all

<code>sort</code>	specifies how and whether variables are sorted in order of the proportion of positives when <code>which="categorical"</code> . Specify <code>sort="none"</code> to leave variables in the order they appear in the original data.
<code>n.unique</code>	the minimum number of distinct values a numeric variable must have before <code>plot.describe</code> uses it in a continuous variable plot
<code>bvspace</code>	the between-variable spacing for categorical variables. Defaults to 2, meaning twice the amount of vertical space as what is used for between-category spacing within a variable
<code>condense</code>	specifies whether to condense the output with regard to the 5 lowest and highest values ("extremes") and the frequency table
<code>lang</code>	specifies the markup language
<code>verb</code>	set to 1 if a verbatim environment is already in effect for LaTeX

### Details

If `options(na.detail.response=TRUE)` has been set and `na.action` is "`na.delete`" or "`na.keep`", summary statistics on the response variable are printed separately for missing and non-missing values of each predictor. The default summary function returns the number of non-missing response values and the mean of the last column of the response values, with a `names` attribute of `c("N", "Mean")`. When the response is a `Surv` object and the mean is used, this will result in the crude proportion of events being used to summarize the response. The actual summary function can be designated through `options(na.fun.response = "function name")`.

If you are modifying LaTeX `par`skip or certain other parameters, you may need to shrink the area around `tabular` and `verbatim` environments produced by `latex.describe`. You can do this using for example `\usepackage{etoolbox}\makeatletter\preto{@verbatim}{\topsep=-1.4pt \partopsep=0pt}\preto{\parsep=0pt}\makeatother` in the LaTeX preamble.

### Value

a list containing elements `descript`, `counts`, `values`. The list is of class `describe`. If the input object was a matrix or a data frame, the list is a list of lists, one list for each variable analyzed. `latex` returns a standard `latex` object. For numeric variables having at least 20 distinct values, an additional component `intervalFreq`. This component is a list with two elements, `range` (containing two values) and `count`, a vector of 100 integer frequency counts. `print` with `which=` returns a 'gt' table object. The user can modify the table by piping formatting changes, column removals, and other operations, before final rendering.

### Author(s)

Frank Harrell  
Vanderbilt University  
<fh@fharrell.com>

### See Also

[spikecomp](#), [sas.get](#), [quantile](#), [GiniMd](#), [pMedian](#), [table](#), [summary](#), [model.frame.default](#), [naprint](#), [lapply](#), [tapply](#), [Surv](#), [na.delete](#), [na.keep](#), [na.detail.response](#), [latex](#)

## Examples

```

set.seed(1)
describe(runif(200),dig=2)    #single variable, continuous
                             #get quantiles .05,.10,\dots

dfr <- data.frame(x=rnorm(400),y=sample(c('male','female'),400,TRUE))
describe(dfr)

## Not run:
options(grType='plotly')
d <- describe(mydata)
p <- plot(d) # create plots for both types of variables
p[[1]]; p[[2]] # or p$Categorical; p$Continuous
plotly::subplot(p[[1]], p[[2]], nrows=2) # plot both in one
plot(d, which='categorical') # categorical ones

d <- sas.get(".", "mydata", special.miss=TRUE, recode=TRUE)
describe(d) #describe entire data frame
attach(d, 1)
describe(relig) #Has special missing values .D .F .M .R .T
                #attr(relig,"label") is "Religious preference"

#relig : Religious preference Format:relig
# n missing D F M R T distinct
# 4038      263 45 33 7 2 1      8
#
#0:none (251, 6%), 1:Jewish (372, 9%), 2:Catholic (1230, 30%)
#3:Jehovah's Witnes (25, 1%), 4:Christ Scientist (7, 0%)
#5:Seventh Day Adv (17, 0%), 6:Protestant (2025, 50%), 7:other (111, 3%)

# Method for describing part of a data frame:
describe(death.time ~ age*sex + rcs(blood.pressure))
describe(~ age+sex)
describe(~ age+sex, weights=freqs) # weighted analysis

fit <- lrm(y ~ age*sex + log(height))
describe(formula(fit))
describe(y ~ age*sex, na.action=na.delete)
# report on number deleted for each variable
options(na.detail.response=TRUE)
# keep missings separately for each x, report on dist of y by x=NA
describe(y ~ age*sex)
options(na.fun.response="quantile")
describe(y ~ age*sex) # same but use quantiles of y by x=NA

d <- describe(my.data.frame)
d$age # print description for just age
d[c('age','sex')] # print description for two variables
d[sort(names(d))] # print in alphabetic order by var. names
d2 <- d[20:30] # keep variables 20-30
page(d2) # pop-up window for these variables

```

```

# Test date/time formats and suppression of times when they don't vary
library(chron)
d <- data.frame(a=chron((1:20)+.1),
                b=chron((1:20)+(1:20)/100),
                d=ISOdatetime(year=rep(2003,20),month=rep(4,20),day=1:20,
                               hour=rep(11,20),min=rep(17,20),sec=rep(11,20)),
                f=ISOdatetime(year=rep(2003,20),month=rep(4,20),day=1:20,
                               hour=1:20,min=1:20,sec=1:20),
                g=ISOdate(year=2001:2020,month=rep(3,20),day=1:20))

describe(d)

# Make a function to run describe, latex.describe, and use the kdvi
# previewer in Linux to view the result and easily make a pdf file

ldesc <- function(data) {
  options(xdviCmd='kdvi')
  d <- describe(data, desc=deparse(substitute(data)))
  dvi(latex(d, file='/tmp/z.tex'), nomargins=FALSE, width=8.5, height=11)
}

ldesc(d)

## End(Not run)

```

---

discrete

*Discrete Vector tools*


---

### Description

discrete creates a discrete vector which is distinct from a continuous vector, or a factor/ordered vector. The other function are tools for manipulating discrete vectors.

### Usage

```

as.discrete(x, ...)
## Default S3 method:
as.discrete(x, ...)
discrete(x, levels = sort(unique.default(x), na.last = TRUE), exclude = NA)
## S3 replacement method for class 'discrete'
x[...] <- value
## S3 method for class 'discrete'
x[..., drop = FALSE]
## S3 method for class 'discrete'
x[[i]]
is.discrete(x)
## S3 replacement method for class 'discrete'
is.na(x) <- value
## S3 replacement method for class 'discrete'
length(x) <- value

```

**Arguments**

x	a vector
drop	Should unused levels be dropped.
exclude	logical: should NA be excluded.
i	indexing vector
levels	character: list of individual level values
value	index of elements to set to NA
...	arguments to be passed to other functions

**Details**

as.discrete converts a vector into a discrete vector.

discrete creates a discrete vector from provided values.

is.discrete tests to see if the vector is a discrete vector.

**Value**

as.discrete, discrete returns a vector of discrete type.

is.discrete returns logical TRUE if the vector is of class discrete otherwise it returns FALSE.

**Author(s)**

Charles Dupont

**See Also**

[\[\[](#), [\[](#), [factor](#)

**Examples**

```
a <- discrete(1:25)
a

is.discrete(a)

b <- as.discrete(2:4)
b
```

dotchart2

*Enhanced Dot Chart***Description**

dotchart2 is an enhanced version of the dotchart function with several new options.

**Usage**

```
dotchart2(data, labels, groups=NULL, gdata=NA, horizontal=TRUE, pch=16,
          xlab='', ylab='', xlim=NULL, auxdata, auxgdata=NULL, auxtitle,
          lty=1, lines=TRUE, dotsize = .8,
          cex = par("cex"), cex.labels = cex,
          cex.group.labels = cex.labels*1.25, sort.=TRUE,
          add=FALSE, dotfont=par('font'), groupfont=2,
          reset.par=add, xaxis=TRUE, width.factor=1.1,
          lcolor='gray', leavepar=FALSE,
          axisat=NULL, axislabels=NULL, ...)
```

**Arguments**

data	a numeric vector whose values are shown on the x-axis
labels	a vector of labels for each point, corresponding to x. If omitted, names(data) are used, and if there are no names, integers prefixed by "#" are used.
groups	an optional categorical variable indicating how data values are grouped
gdata	data values for groups, typically summaries such as group medians
horizontal	set to FALSE to make the chart vertical instead of the default
pch	default character number or value for plotting dots in dot charts. The default is 16.
xlab	x-axis title
ylab	y-axis title
xlim	x-axis limits. Applies only to horizontal=TRUE.
auxdata	a vector of auxiliary data given to dotchart2, of the same length as the first (data) argument. If present, this vector of values will be printed outside the right margin of the dot chart. Usually auxdata represents cell sizes.
auxgdata	similar to auxdata but corresponding to the gdata argument. These usually represent overall sample sizes for each group of lines.
auxtitle	if auxdata is given, auxtitle specifies a column heading for the extra printed data in the chart, e.g., "N"
lty	line type for horizontal lines. Default is 1 for R, 2 for S-Plus
lines	set to FALSE to suppress drawing of reference lines
dotsize	cex value for drawing dots. Default is 0.8. Note that the original dotchart function used a default of 1.2.

<code>cex</code>	see <a href="#">par</a>
<code>cex.labels</code>	<code>cex</code> parameter that applies only to the line labels for the dot chart <code>cex</code> parameter for major grouping labels for <code>dotchart2</code> . Defaults to <code>cex</code> .
<code>cex.group.labels</code>	value of <code>cex</code> corresponding to <code>gdata</code>
<code>sort.</code>	set to <code>FALSE</code> to keep <code>dotchart2</code> from sorting the input data, i.e., it will assume that the data are already properly arranged. This is especially useful when you are using <code>gdata</code> and <code>groups</code> and you want to control the order that groups appear on the chart (from top to bottom).
<code>add</code>	set to <code>TRUE</code> to add to an existing plot
<code>dotfont</code>	font number of plotting dots. Default is one. Use <code>-1</code> to use "outline" fonts. For example, <code>pch=183</code> , <code>dotfont=-1</code> plots an open circle for UNIX on postscript. <code>pch=1</code> makes an open octagon under Windows.
<code>groupfont</code>	font number to use in drawing group labels for <code>dotchart2</code> . Default is 2 for boldface.
<code>reset.par</code>	set to <code>FALSE</code> to cause <code>dotchart2</code> to not reset the <code>par</code> parameters when finished. This is useful when <code>add=TRUE</code> is about to be used in another call. The default is to reset the <code>par</code> parameters if <code>add=TRUE</code> and not if <code>add=FALSE</code> , i.e., the program assumes that only one set of points will be added to an existing set. If you fail to use <code>reset.par=TRUE</code> for the first of a series of plots, the next call to <code>plot</code> with <code>add=TRUE</code> will result in distorted x-axis scaling.
<code>xaxis</code>	set to <code>FALSE</code> to suppress drawing x-axis
<code>width.factor</code>	When the calculated left margin turns out to be faulty, specify a factor by which to multiple the left margin as <code>width.factor</code> to get the appropriate space for labels on horizontal charts.
<code>lcolor</code>	color for horizontal reference lines. Default is "gray" for R, <code>par("col")</code> for S-Plus.
<code>leavepar</code>	set to <code>TRUE</code> to leave <code>par()</code> unchanged. This assumes the user has allocated sufficient left and right margins for a horizontal dot chart.
<code>axisat</code>	a vector of tick mark locations to pass to <code>axis</code> . Useful if transforming the data axis
<code>axislabels</code>	a vector of strings specifying axis tick mark labels. Useful if transforming the data axis
<code>...</code>	arguments passed to <code>plot.default</code>

### Side Effects

`dotchart` will leave `par` altered if `reset.par=FALSE`.

### Author(s)

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>



**See Also**[dotchart](#)**Examples**

```

set.seed(135)
maj <- factor(c(rep('North',13),rep('South',13)))
g <- paste('Category',rep(letters[1:13],2))
n <- sample(1:15000, 26, replace=TRUE)
y1 <- runif(26)
y2 <- pmax(0, y1 - runif(26, 0, .1))
dotchart2(y1, g, groups=maj, auxdata=n, auxtitle='n', xlab='Y')
dotchart2(y2, g, groups=maj, pch=17, add=TRUE)
## Compare with dotchart function (no superpositioning or auxdata allowed):
## dotchart(y1, g, groups=maj, xlab='Y')

## To plot using a transformed scale add for example
## axisat=sqrt(pretty(y)), axislabels=pretty(y)

```

dotchart3

*Enhanced Version of dotchart Function***Description**

These are adaptations of the R dotchart function that sorts categories top to bottom, adds auxdata and auxtitle arguments to put extra information in the right margin, and for dotchart3 adds arguments cex.labels, cex.group.labels, and groupfont. By default, group headings are in a larger, bold font. dotchart3 also cuts a bit of white space from the top and bottom of the chart. The most significant change, however, is in how x is interpreted. Columns of x no longer provide an alternate way to define groups. Instead, they define superpositioned values. This is useful for showing three quartiles, for example. Going along with this change, for dotchart3 pch can now be a vector specifying symbols to use going across columns of x. x was changed in this way because to put multiple points on a line (e.g., quartiles) and keeping track of par() parameters when dotchart2 was called with add=TRUE was cumbersome. dotchart3 changes the margins to account for horizontal labels.

dotchartp is a version of dotchart3 for making the chart with the plotly package.

summaryD creates aggregate data using [summarize](#) and calls dotchart3 with suitable arguments to summarize data by major and minor categories. If options(grType='plotly') is in effect and the plotly package is installed, summaryD uses dotchartp instead of dotchart3.

summaryDp is a streamlined summaryD-like function that uses the dotchartp1 function to render a plotly graphic. It is used to compute summary statistics stratified separately by a series of variables.

**Usage**

```
dotchart3(x, labels = NULL, groups = NULL, gdata = NULL,
          cex = par("cex"), pch = 21, gpch = pch, bg = par("bg"),
          color = par("fg"), gcolor = par("fg"), lcolor = "gray",
          xlim = range(c(x, gdata), na.rm=TRUE), main = NULL, xlab = NULL,
          ylab = NULL, auxdata = NULL, auxtitle = NULL, auxgdata=NULL,
          axisat=NULL, axislabels=NULL,
          cex.labels = cex, cex.group.labels = cex.labels * 1.25,
          cex.auxdata=cex, groupfont = 2,
          auxwhere=NULL, height=NULL, width=NULL, ...)
```

```
dotchartp(x, labels = NULL, groups = NULL, gdata = NULL,
          xlim = range(c(x, gdata), na.rm=TRUE), main=NULL,
          xlab = NULL, ylab = '', auxdata=NULL, auxtitle=NULL,
          auxgdata=NULL, auxwhere=c('right', 'hover'),
          symbol='circle', col=colspace::rainbow_hcl,
          legendgroup=NULL,
          axisat=NULL, axislabels=NULL, sort=TRUE, digits=4, dec=NULL,
          height=NULL, width=700, layoutattr=FALSE, showlegend=TRUE, ...)
```

```
summaryD(formula, data=NULL, fun=mean, funm=fun,
          groupsummary=TRUE, auxvar=NULL, auxtitle='',
          auxwhere=c('hover', 'right'),
          vals=length(auxvar) > 0, fmtvals=format,
          symbol=if(use.plotly) 'circle' else 21,
          col=if(use.plotly) colorspace::rainbow_hcl else 1:10,
          legendgroup=NULL,
          cex.auxdata=.7, xlab=v[1], ylab=NULL,
          grillevery=NULL, gridcol=gray(.95), sort=TRUE, ...)
```

```
summaryDp(formula,
           fun=function(x) c(Mean=mean(x, na.rm=TRUE),
                             N=sum(! is.na(x))),
           overall=TRUE, xlim=NULL, xlab=NULL,
           data=NULL, subset=NULL, na.action=na.retain,
           ncharsmax=c(50, 30),
           digits=4, ...)
```

**Arguments**

x	a numeric vector or matrix
labels	labels for categories corresponding to rows of x. If not specified these are taken from row names of x.
groups, gdata, cex, pch, gpch, bg, color, gcolor, lcolor, xlim, main, xlab, ylab	see <a href="#">dotchart</a>
auxdata	a vector of information to be put in the right margin, in the same order as x. May be numeric, character, or a vector of expressions containing <a href="#">plotmath</a> markup.

	For dotchartp, auxdata may be a matrix to go along with the numeric x-axis variable, to result in point-specific hover text.
auxtitle	a column heading for auxdata
auxgdata	similar to auxdata but corresponding to the gdata argument. These usually represent overall sample sizes for each group of lines.
axisat	a vector of tick mark locations to pass to axis. Useful if transforming the data axis
axislabels	a vector of strings specifying axis tick mark labels. Useful if transforming the data axis
digits	number of significant digits for formatting numeric data in hover text for dotchartp and summaryDp
dec	for dotchartp only, overrides digits to specify the argument to round() for rounding values for hover labels
cex.labels	cex for labels
cex.group.labels	cex for group labels
cex.auxdata	cex for auxdata
groupfont	font number for group headings
auxwhere	for summaryD and dotchartp specifies whether auxdata and auxgdata are to be placed on the far right of the chart, or should appear as pop-up tooltips when hovering the mouse over the ordinary x data points on the chart. Ignored for dotchart3.
...	other arguments passed to some of the graphics functions, or to dotchart3 or dotchartp from summaryD. The auxwhere='hover' option is a useful argument to pass from summaryD to dotchartp. Also used to pass other arguments to dotchartpl from summaryDp.
layoutattr	set to TRUE to put plotly::layout information in a list as an attribute layout of the returned plotly object instead of running the plotly object through the layout function. This is useful if running dotchartp multiple times to later put together using plotly::subplot and only then running the result through plotly::layout.
showlegend	set to FALSE to suppress the plotly legend with dotchartp
formula	a formula with one variable on the left hand side (the variable to compute summary statistics on), and one or two variables on the right hand side. If there are two variables, the first is taken as the major grouping variable. If the left hand side variable is a matrix it has to be a legal R variable name, not an expression, and fun needs to be able to process a matrix. For summaryDp there may be more than two right-hand-side variables.
data	a data frame or list used to find the variables in formula. If omitted, the parent environment is used.
fun	a summarization function creating a single number from a vector. Default is the mean. For summaryDp, fun produces a named vector of summary statistics, with the default computing the Mean and N (number of non-missing values).

funm	applies if there are two right hand variables and groupsummary=TRUE and the marginal summaries over just the first x variable need to be computed differently than the summaries that are cross-classified by both variables. funm defaults to fun and should have the same structure as fun.
groupsummary	By default, when there are two right-hand variables, summarize(..., fun) is called a second time without the use of the second variable, to obtain marginal summaries for the major grouping variable and display the results as a dot (and optionally in the right margin). Set groupsummary=FALSE to suppress this information.
auxvar	when fun returns more than one statistic and the user names the elements in the returned vector, you can specify auxvar as a single character string naming one of them. This will cause the named element to be written in the right margin, and that element to be deleted when plotting the statistics.
vals	set to TRUE to show data values (dot locations) in the right margin. Defaults to TRUE if auxvar is specified.
fmtvals	an optional function to format values before putting them in the right margin. Default is the format function.
symbol	a scalar or vector of pch values for ordinary graphics or a character vector or scalar of plotly symbols. These correspond to columns of x or elements produced by fun.
col	a function or vector of colors to assign to multiple points plotted in one line. If a function it will be evaluated with an argument equal to the number of groups/columns.
legendgroup	see plotly documentation; corresponds to column names/fun output for plotly graphs only
gridevery	specify a positive number to draw very faint vertical grid lines every gridevery x-axis units; for non-plotly charts
gridcol	color for grid lines; default is very faint gray scale
sort	specify sort=FALSE to plot data in the original order, from top to bottom on the dot chart. For dotchartp, set sort to 'descending' to sort in descending order of the first column of x, or 'ascending' to do the reverse. These do not make sense if groups is present.
height, width	height and width in pixels for dotchartp if not using plotly defaults. Ignored for dotchart3. If set to "auto" the height is computed using Hmisc::plotlyHeightDotchart.
overall	set to FALSE to suppress plotting of unstratified estimates
subset	an observation subsetting expression
na.action	an NA action function
ncharsmax	a 2-vector specifying the number of characters after which an html new line character should be placed, respectively for the x-axis label and the stratification variable levels

### Value

the function returns invisibly

**Author(s)**

Frank Harrell

**See Also**[dotchart](#), [dotchart2](#), [summarize](#), [rlegend](#)**Examples**

```

set.seed(135)
maj <- factor(c(rep('North',13),rep('South',13)))
g <- paste('Category',rep(letters[1:13],2))
n <- sample(1:15000, 26, replace=TRUE)
y1 <- runif(26)
y2 <- pmax(0, y1 - runif(26, 0, .1))
dotchart3(cbind(y1,y2), g, groups=maj, auxdata=n, auxtitle='n',
          xlab='Y', pch=c(1,17))
## Compare with dotchart function (no superpositioning or auxdata allowed):
## dotchart(y1, g, groups=maj, xlab='Y')

## Not run:
dotchartp(cbind(y1, y2), g, groups=maj, auxdata=n, auxtitle='n',
          xlab='Y', gdata=cbind(c(0,.1), c(.23,.44)), auxgdata=c(-1,-2),
          symbol=c('circle', 'line-ns-open'))

summaryDp(sbp ~ region + sex + race + cut2(age, g=5), data=mydata)

## End(Not run)

## Put options(grType='plotly') to have the following use dotchartp
## (rlegend will not apply)
## Add argument auxwhere='hover' to summaryD or dotchartp to put
## aux info in hover text instead of right margin
summaryD(y1 ~ maj + g, xlab='Mean')
summaryD(y1 ~ maj + g, groupsummary=FALSE)
summaryD(y1 ~ g, fmtvals=function(x) sprintf('%4.2f', x))
Y <- cbind(y1, y2) # summaryD cannot handle cbind(...) ~ ...
summaryD(Y ~ maj + g, fun=function(y) y[1,], symbol=c(1,17))
rlegend(.1, 26, c('y1','y2'), pch=c(1,17))

summaryD(y1 ~ maj, fun=function(y) c(Mean=mean(y), n=length(y)),
          auxvar='n', auxtitle='N')

```

## Description

This function produces a plotly interactive graphic and accepts a different format of data input than the other dotchart functions. It was written to handle a hierarchical data structure including strata that further subdivide the main classes. Strata, indicated by the `mult` variable, are shown on the same horizontal line, and if the variable `big` is `FALSE` will appear slightly below the main line, using smaller symbols, and having some transparency. This is intended to handle output such as that from the `summaryP` function when there is a superpositioning variable group and a stratification variable `mult`, especially when the data have been run through the `addMarginal` function to create `mult` categories labelled "All" for which the user will specify `big=TRUE` to indicate non-stratified estimates (stratified only on group) to emphasize.

When viewing graphics that used `mult` and `big`, the user can click on the legends for the small points for groups to vanish the finely stratified estimates.

When `group` is used by `mult` and `big` are not, and when the `group` variable has exactly two distinct values, you can specify `refgroup` to get the difference between two proportions in addition to the individual proportions. The individual proportions are plotted, but confidence intervals for the difference are shown in hover text and half-width confidence intervals for the difference, centered at the midpoint of the proportions, are shown. These have the property of intersecting the two proportions if and only if there is no significant difference at the  $1 - \text{conf.int}$  level.

Specify `fun=exp` and `ifun=log` if estimates and confidence limits are on the log scale. Make sure that zeros were prevented in the original calculations. For exponential hazard rates this can be accomplished by replacing event counts of 0 with 0.5.

## Usage

```
dotchartpl(x, major=NULL, minor=NULL, group=NULL, mult=NULL,
           big=NULL, htext=NULL, num=NULL, denom=NULL,
           numlabel='', denomlabel='',
           fun=function(x) x, ifun=function(x) x, op='-',
           lower=NULL, upper=NULL,
           refgroup=NULL, sortdiff=TRUE, conf.int=0.95,
           minkeep=NULL, xlim=NULL, xlab='Proportion',
           tracename=NULL, limitstracename='Limits',
           nonbigtracename='Stratified Estimates',
           dec=3, width=800, height=NULL,
           col=colorspace::rainbow_hcl)
```

## Arguments

<code>x</code>	a numeric vector used for values on the x-axis
<code>major</code>	major vertical category, e.g., variable labels
<code>minor</code>	minor vertical category, e.g. category levels within variables
<code>group</code>	superpositioning variable such as treatment
<code>mult</code>	strata names for further subdivisions without groups
<code>big</code>	omit if all levels of <code>mult</code> are equally important or if <code>mult</code> is omitted. Otherwise denotes major (larger points, right on horizontal lines) vs. minor (smaller, transparent points slightly below the line).

h <code>text</code>	additional hover text per point
<code>num</code>	if <code>x</code> represents proportions, optionally specifies numerators to be used in fractions added to hover text. When <code>num</code> is given, <code>x</code> is automatically added to hover text, rounded to 3 digits after the decimal point.
<code>denom</code>	like <code>num</code> but for denominators
<code>numlabel</code>	character string to put to the right of the numerator in hover text
<code>denomlabel</code>	character string to put to the right of the denominator in hover text
<code>fun</code>	a transformation to make when printing estimates. For example, one may specify <code>fun=exp</code> to anti-log estimates and confidence limites that were computed on a log basis
<code>ifun</code>	inverse transformation of <code>fun</code>
<code>op</code>	set to for example <code>'/'</code> when <code>fun=exp</code> and effects are computed as ratios instead of differences. This is used in hover text.
<code>lower</code>	lower limits for optional error bars
<code>upper</code>	upper limits for optional error bars
<code>refgroup</code>	if <code>group</code> is specified and there are exactly two groups, specify the character string for the reference group in computing difference in proportions. For example if <code>refgroup='A'</code> and the group levels are <code>'A'</code> , <code>'B'</code> , you will get <code>B - A</code> .
<code>sortdiff</code>	minor categories are sorted by descending values of the difference in proportions when <code>refgroup</code> is used, unless you specify <code>sortdiff=FALSE</code>
<code>conf.int</code>	confidence level for computing confidence intervals for the difference in two proportions. Specify <code>conf.int=FALSE</code> to suppress confidence intervals.
<code>minkeep</code>	if <code>refgroup</code> and <code>minkeep</code> are both given, observations that are at or above <code>minkeep</code> for at least one of the groups are retained. The defaults to to keep all observations.
<code>xlim</code>	x-axis limits
<code>xlab</code>	x-axis label
<code>tracename</code>	plotly trace name if <code>group</code> is not used
<code>limitstracename</code>	plotly trace name for lower and upper if <code>group</code> is not used
<code>nonbigtracename</code>	plotly trace name used for non-big elements, which usually represent stratified versions of the "big" observations
<code>col</code>	a function or vector of colors to assign to <code>group</code> . If a function it will be evaluated with an argument equal to the number of distinct groups.
<code>dec</code>	number of places to the right of the decimal place for formatting numeric quantities in hover text
<code>width</code>	width of plot in pixels
<code>height</code>	height of plot in pixels; computed from number of strata by default

**Value**

a plotly object. An attribute `levelsRemoved` is added if `minkeep` is used and any categories were omitted from the plot as a result. This is a character vector with categories removed. If `major` is present, the strings are of the form `major:minor`

**Author(s)**

Frank Harrell

**See Also**

[dotchartp](#)

**Examples**

```
## Not run:
set.seed(1)
d <- expand.grid(major=c('Alabama', 'Alaska', 'Arkansas'),
                minor=c('East', 'West'),
                group=c('Female', 'Male'),
                city=0:2)

n <- nrow(d)
d$num <- round(100*runif(n))
d$denom <- d$num + round(100*runif(n))
d$x <- d$num / d$denom
d$lower <- d$x - runif(n)
d$upper <- d$x + runif(n)

with(d,
  dotchartpl(x, major, minor, group, city, lower=lower, upper=upper,
             big=city==0, num=num, denom=denom, xlab='x'))

# Show half-width confidence intervals for Female - Male differences
# after subsetting the data to have only one record per
# state/region/group
d <- subset(d, city == 0)
with(d,
  dotchartpl(x, major, minor, group, num=num, denom=denom,
             lower=lower, upper=upper, refgroup='Male')
)

n <- 500
set.seed(1)
d <- data.frame(
  race      = sample(c('Asian', 'Black/AA', 'White'), n, TRUE),
  sex       = sample(c('Female', 'Male'), n, TRUE),
  treat     = sample(c('A', 'B'), n, TRUE),
  smoking   = sample(c('Smoker', 'Non-smoker'), n, TRUE),
  hypertension = sample(c('Hypertensive', 'Non-Hypertensive'), n, TRUE),
  region    = sample(c('North America', 'Europe', 'South America',
                     'Europe', 'Asia', 'Central America'), n, TRUE))
```



```

d <- upData(d, labels=c(race='Race', sex='Sex'))

dm <- addMarginal(d, region)
s <- summaryP(race + sex + smoking + hypertension ~
              region + treat, data=dm)

s$region <- ifelse(s$region == 'All', 'All Regions', as.character(s$region))

with(s,
  dotchartpl(freq / denom, major=var, minor=val, group=treat, mult=region,
             big=region == 'All Regions', num=freq, denom=denom)
)

s2 <- s[- attr(s, 'rows.to.exclude1'), ]
with(s2,
  dotchartpl(freq / denom, major=var, minor=val, group=treat, mult=region,
             big=region == 'All Regions', num=freq, denom=denom)
)
# Note these plots can be created by plot.summaryP when options(grType='plotly')

# Plot hazard rates and ratios with confidence limits, on log scale
d <- data.frame(tx=c('a', 'a', 'b', 'b'),
               event=c('MI', 'stroke', 'MI', 'stroke'),
               count=c(10, 5, 5, 2),
               exposure=c(1000, 1000, 900, 900))
# There were no zero event counts in this dataset. In general we
# want to handle that, hence the 0.5 below
d <- upData(d, hazard = pmax(0.5, count) / exposure,
           selog = sqrt(1. / pmax(0.5, count)),
           lower = log(hazard) - 1.96 * selog,
           upper = log(hazard) + 1.96 * selog)

with(d,
  dotchartpl(log(hazard), minor=event, group=tx, num=count, denom=exposure,
             lower=lower, upper=upper,
             fun=exp, ifun=log, op='/',
             numlabel='events', denomlabel='years',
             refgroup='a', xlab='Events Per Person-Year')
)

## End(Not run)

```

---

ebpcomp

*ebpcomp*


---

## Description

Computation of Coordinates of Extended Box Plots Elements

## Usage

```
ebpcomp(x, qref = c(0.5, 0.25, 0.75), probs = c(0.05, 0.125, 0.25, 0.375))
```

**Arguments**

x	a numeric variable
qref	quantiles for major corners
probs	quantiles for minor corners

**Details**

For an extended box plots computes all the elements needed for plotting it. This is typically used when adding to a ggplot2 plot.

**Value**

list with elements segments, lines, points, points2

**Author(s)**

Frank Harrell

**Examples**

```
ebpcomp(1:1000)
```

---

Ecdf

*Empirical Cumulative Distribution Plot*

---

**Description**

Computes coordinates of cumulative distribution function of x, and by defaults plots it as a step function. A grouping variable may be specified so that stratified estimates are computed and (by default) plotted. If there is more than one group, the labcurve function is used (by default) to label the multiple step functions or to draw a legend defining line types, colors, or symbols by linking them with group labels. A weights vector may be specified to get weighted estimates. Specify normwt to make weights sum to the length of x (after removing NAs). Other wise the total sample size is taken to be the sum of the weights.

Ecdf is actually a method, and Ecdf.default is what's called for a vector argument. Ecdf.data.frame is called when the first argument is a data frame. This function can automatically set up a matrix of ECDFs and wait for a mouse click if the matrix requires more than one page. Categorical variables, character variables, and variables having fewer than a set number of unique values are ignored. If par(mfrow=. .) is not set up before Ecdf.data.frame is called, the function will try to figure the best layout depending on the number of variables in the data frame. Upon return the original mfrow is left intact.

When the first argument to Ecdf is a formula, a Trellis/Lattice function Ecdf.formula is called. This allows for multi-panel conditioning, superposition using a groups variable, and other Trellis features, along with the ability to easily plot transformed ECDFs using the fun argument. For example, if fun=qnorm, the inverse normal transformation will be used for the y-axis. If the transformed curves are linear this indicates normality. Like the xYplot function, Ecdf will create a function Key if the groups variable is used. This function can be invoked by the user to define the keys for the groups.

**Usage**

```

Ecdf(x, ...)

## Default S3 method:
Ecdf(x, what=c('F','1-F','f','1-f'),
      weights=rep(1, length(x)), normwt=FALSE,
      xlab, ylab, q, pl=TRUE, add=FALSE, lty=1,
      col=1, group=rep(1,length(x)), label.curves=TRUE, xlim,
      subtitles=TRUE, datadensity=c('none','rug','hist','density'),
      side=1,
      frac=switch(datadensity,none=NA,rug=.03,hist=.1,density=.1),
      dens.opts=NULL, lwd=1, log='', ...)

## S3 method for class 'data.frame'
Ecdf(x, group=rep(1,nrows),
      weights=rep(1, nrows), normwt=FALSE,
      label.curves=TRUE, n.unique=10, na.big=FALSE, subtitles=TRUE,
      vnames=c('labels','names'),...)

## S3 method for class 'formula'
Ecdf(x, data=sys.frame(sys.parent()), groups=NULL,
      prepanel=prepanel.Ecdf, panel=panel.Ecdf, ..., xlab,
      ylab, fun=function(x)x, what=c('F','1-F','f','1-f'), subset=TRUE)

```

**Arguments**

x	a numeric vector, data frame, or Trellis/Lattice formula
what	The default is "F" which results in plotting the fraction of values $\leq x$ . Set to "1-F" to plot the fraction $> x$ or "f" to plot the cumulative frequency of values $\leq x$ . Use "1-f" to plot the cumulative frequency of values $\geq x$ .
weights	numeric vector of weights. Omit or specify a zero-length vector or NULL to get unweighted estimates.
normwt	see above
xlab	x-axis label. Default is label(x) or name of calling argument. For Ecdf . formula, xlab defaults to the label attribute of the x-axis variable.
ylab	y-axis label. Default is "Proportion $\leq x$ ", "Proportion $> x$ ", or "Frequency $\leq x$ " depending on value of what.
q	a vector for quantiles for which to draw reference lines on the plot. Default is not to draw any.
pl	set to F to omit the plot, to just return estimates
add	set to TRUE to add the cdf to an existing plot. Does not apply if using lattice graphics (i.e., if a formula is given as the first argument).
lty	integer line type for plot. If group is specified, this can be a vector.
lwd	line width for plot. Can be a vector corresponding to groups.

log	see <a href="#">plot</a> . Set log='x' to use log scale for x-axis.
col	color for step function. Can be a vector.
group	a numeric, character, or factor categorical variable used for stratifying estimates. If group is present, as many ECDFs are drawn as there are non-missing group levels.
label.curves	applies if more than one group exists. Default is TRUE to use labcurve to label curves where they are farthest apart. Set label.curves to a list to specify options to labcurve, e.g., label.curves=list(method="arrow", cex=.8). These option names may be abbreviated in the usual way arguments are abbreviated. Use for example label.curves=list(keys=1:5) to draw symbols periodically (as in pch=1:5 - see points) on the curves and automatically position a legend in the most empty part of the plot. Set label.curves=FALSE to suppress drawing curve labels. The col, lty, and type parameters are automatically passed to labcurve, although you can override them here. You can set label.curves=list(keys="lines") to have different line types defined in an automatically positioned key.
xlim	x-axis limits. Default is entire range of x.
subtitles	set to FALSE to suppress putting a subtitle at the bottom left of each plot. The subtitle indicates the numbers of non-missing and missing observations, which are labeled n, m.
datadensity	If datadensity is not "none", either scat1d or histSpike is called to add a rug plot (datadensity="rug"), spike histogram (datadensity="hist"), or smooth density estimate ("density") to the bottom or top of the ECDF.
side	If datadensity is not "none", the default is to place the additional information on top of the x-axis (side=1). Use side=3 to place at the top of the graph.
frac	passed to histSpike
dens.opts	a list of optional arguments for histSpike
...	other parameters passed to plot if add=F. For data frames, other parameters to pass to Ecdf.default. For Ecdf.formula, if groups is not used, you can also add data density information to each panel's ECDF by specifying the datadensity and optional frac, side, dens.opts arguments.
n.unique	minimum number of unique values before an ECDF is drawn for a variable in a data frame. Default is 10.
na.big	set to TRUE to draw the number of NAs in larger letters in the middle of the plot for Ecdf.data.frame
vnames	By default, variable labels are used to label x-axes. Set vnames="names" to instead use variable names.
method	method for computing the empirical cumulative distribution. See wtd.Ecdf. The default is to use the standard "i/n" method as is used by the non-Trellis versions of Ecdf.
fun	a function to transform the cumulative proportions, for the Trellis-type usage of Ecdf
data, groups, subset, prepanel, panel	the usual Trellis/Lattice parameters, with groups causing Ecdf.formula to overlay multiple ECDFs on one panel.

**Value**

for `Ecdf.default` an invisible list with elements `x` and `y` giving the coordinates of the cdf. If there is more than one group, a list of such lists is returned. An attribute, `N`, is in the returned object. It contains the elements `n` and `m`, the number of non-missing and missing observations, respectively.

**Side Effects**

plots

**Author(s)**

Frank Harrell  
Department of Biostatistics, Vanderbilt University  
<fh@fharrell.com>

**See Also**

[wtd.Ecdf](#), [label](#), [table](#), [cumsum](#), [labcurve](#), [xYplot](#), [histSpike](#)

**Examples**

```
set.seed(1)
ch <- rnorm(1000, 200, 40)
Ecdf(ch, xlab="Serum Cholesterol")
scat1d(ch) # add rug plot
histSpike(ch, add=TRUE, frac=.15) # add spike histogram
# Better: add a data density display automatically:
Ecdf(ch, datadensity='density')

label(ch) <- "Serum Cholesterol"
Ecdf(ch)
other.ch <- rnorm(500, 220, 20)
Ecdf(other.ch, add=TRUE, lty=2)

sex <- factor(sample(c('female', 'male'), 1000, TRUE))
Ecdf(ch, q=c(.25, .5, .75)) # show quartiles
Ecdf(ch, group=sex,
      label.curves=list(method='arrow'))

# Example showing how to draw multiple ECDFs from paired data
pre.test <- rnorm(100, 50, 10)
post.test <- rnorm(100, 55, 10)
x <- c(pre.test, post.test)
g <- c(rep('Pre', length(pre.test)), rep('Post', length(post.test)))
Ecdf(x, group=g, xlab='Test Results', label.curves=list(keys=1:2))
# keys=1:2 causes symbols to be drawn periodically on top of curves
```

```

# Draw a matrix of ECDFs for a data frame
m <- data.frame(pre.test, post.test,
                sex=sample(c('male','female'),100,TRUE))
Ecdf(m, group=m$sex, datadensity='rug')

freqs <- sample(1:10, 1000, TRUE)
Ecdf(ch, weights=freqs) # weighted estimates

# Trellis/Lattice examples:

region <- factor(sample(c('Europe','USA','Australia'),100,TRUE))
year <- factor(sample(2001:2002,1000,TRUE))
Ecdf(~ch | region*year, groups=sex)
Key() # draw a key for sex at the default location
# Key(locator(1)) # user-specified positioning of key
age <- rnorm(1000, 50, 10)
Ecdf(~ch | lattice::equal.count(age), groups=sex) # use overlapping shingles
Ecdf(~ch | sex, datadensity='hist', side=3) # add spike histogram at top

```

---

ecdfSteps

*ecdfSteps*


---

## Description

Compute Coordinates of an Empirical Distribution Function

## Usage

```
ecdfSteps(x, extend)
```

## Arguments

x	numeric vector, possibly with NAs that are ignored
extend	a 2-vector do extend the range of x (low, high). Set extend=FALSE to not extend x, or leave it missing to extend it 1/20th of the observed range on other side.

## Details

For a numeric vector uses the R built-in `ecdf` function to compute coordinates of the ECDF, with extension slightly below and above the range of x by default. This is useful for `ggplot2` where the ECDF may need to be transformed. The returned object is suitable for creating stratified statistics using `data.table` and other methods.

## Value

a list with components x and y

**Author(s)**

Frank Harrell

**See Also**[stats::ecdf\(\)](#)**Examples**

```
ecdfSteps(0:10)
## Not run:
# Use data.table for obtaining ECDFs by country and region
w <- d[, ecdfSteps(z, extend=c(1,11)), by=.(country, region)] # d is a DT
# Use ggplot2 to make one graph with multiple regions' ECDFs
# and use faceting for countries
ggplot(w, aes(x, y, color=region)) + geom_step() +
  facet_wrap(~ country)

## End(Not run)
```

---

equalBins

*Multicolumn Formating*

---

**Description**

Expands the width either supercolumns or the subcolumns so that the the sum of the supercolumn widths is the same as the sum of the subcolumn widths.

**Usage**

```
equalBins(widths, subwidths)
```

**Arguments**

widths	widths of the supercolumns.
subwidths	list of widths of the subcolumns for each supercolumn.

**Details**

This determines the correct subwidths of each of various columns in a table for printing. The correct width of the multicolumns is determined by summing the widths of it subcolumns.

**Value**

widths of the the columns for a table.

**Author(s)**

Charles Dupont

**See Also**

[nchar](#), [stringDims](#)

**Examples**

```
mcols <- c("Group 1", "Group 2")
mwidth <- nchar(mcols, type="width")
spancols <- c(3,3)
ccols <- c("a", "deer", "ad", "cat", "help", "bob")
cwidth <- nchar(ccols, type="width")

subwidths <- partition.vector(cwidth, spancols)

equalBins(mwidth, subwidths)
```

---

errbar

*Plot Error Bars*


---

**Description**

Add vertical error bars to an existing plot or makes a new plot with error bars.

**Usage**

```
errbar(x, y, yplus, yminus, cap=0.015, main = NULL,
       sub=NULL, xlab=as.character(substitute(x)),
       ylab=if(is.factor(x) || is.character(x)) ""
             else as.character(substitute(y)),
       add=FALSE, lty=1, type='p', ylim=NULL,
       lwd=1, pch=16, errbar.col, Type=rep(1, length(y)),
       ...)
```

**Arguments**

x	vector of numeric x-axis values (for vertical error bars) or a factor or character variable (for horizontal error bars, x representing the group labels)
y	vector of y-axis values.
yplus	vector of y-axis values: the tops of the error bars.
yminus	vector of y-axis values: the bottoms of the error bars.
cap	the width of the little lines at the tops and bottoms of the error bars in units of the width of the plot. Defaults to 0.015.
main	a main title for the plot, passed to plot, see also <a href="#">title</a> .
sub	a sub title for the plot, passed to plot
xlab	optional x-axis labels if add=FALSE.
ylab	optional y-axis labels if add=FALSE. Defaults to blank for horizontal charts.



add	set to TRUE to add bars to an existing plot (available only for vertical error bars)
lty	type of line for error bars
type	type of point. Use type="b" to connect dots.
ylim	y-axis limits. Default is to use range of y, yminus, and yplus. For horizontal charts, ylim is really the x-axis range, excluding differences.
lwd	line width for line segments (not main line)
pch	character to use as the point.
errbar.col	color to use for drawing error bars.
Type	used for horizontal bars only. Is an integer vector with values 1 if corresponding values represent simple estimates, 2 if they represent differences.
...	other parameters passed to all graphics functions.

### Details

errbar adds vertical error bars to an existing plot or makes a new plot with error bars. It can also make a horizontal error bar plot that shows error bars for group differences as well as bars for groups. For the latter type of plot, the lower x-axis scale corresponds to group estimates and the upper scale corresponds to differences. The spacings of the two scales are identical but the scale for differences has its origin shifted so that zero may be included. If at least one of the confidence intervals includes zero, a vertical dotted reference line at zero is drawn.

### Author(s)

Charles Geyer, University of Chicago. Modified by Frank Harrell, Vanderbilt University, to handle missing data, to add the parameters add and lty, and to implement horizontal charts with differences.

### Examples

```
set.seed(1)
x <- 1:10
y <- x + rnorm(10)
delta <- runif(10)
errbar( x, y, y + delta, y - delta )

# Show bootstrap nonparametric CLs for 3 group means and for
# pairwise differences on same graph
group <- sample(c('a','b','d'), 200, TRUE)
y <- runif(200) + .25*(group=='b') + .5*(group=='d')
cla <- smean.cl.boot(y[group=='a'],B=100, reps=TRUE) # usually B=1000
a <- attr(cla, 'reps')
clb <- smean.cl.boot(y[group=='b'],B=100, reps=TRUE)
b <- attr(clb, 'reps')
cld <- smean.cl.boot(y[group=='d'],B=100, reps=TRUE)
d <- attr(cld, 'reps')
a.b <- quantile(a-b,c(.025,.975))
a.d <- quantile(a-d,c(.025,.975))
```

```

b.d <- quantile(b-d,c(.025,.975))
errbar(c('a','b','d','a - b','a - d','b - d'),
       c(c1a[1],c1b[1],c1d[1],c1a[1]-c1b[1],c1a[1]-c1d[1],c1b[1]-c1d[1]),
       c(c1a[3],c1b[3],c1d[3],a.b[2],a.d[2],b.d[2]),
       c(c1a[2],c1b[2],c1d[2],a.b[1],a.d[1],b.d[1]),
       Type=c(1,1,1,2,2,2), xlab='', ylab='')

```

---

escapeRegex	<i>Escapes any characters that would have special meaning in a regular expression.</i>
-------------	--

---

### Description

Escapes any characters that would have special meaning in a regular expression.

### Usage

```

escapeRegex(string)
escapeBS(string)

```

### Arguments

string            string being operated on.

### Details

escapeRegex will escape any characters that would have special meaning in a regular expression. For any string `grep(regexEscape(string), string)` will always be true.

escapeBS will escape any backslash ‘\’ in a string.

### Value

The value of the string with any characters that would have special meaning in a regular expression escaped.

### Author(s)

Charles Dupont  
 Department of Biostatistics  
 Vanderbilt University

### See Also

[grep](#)

**Examples**

```
string <- "this\\(system) {is} [full]."  
escapeRegex(string)  
  
escapeBS(string)
```

---

estSeqMarkovOrd      *estSeqMarkovOrd*

---

**Description**

Simulate Comparisons For Use in Sequential Markov Longitudinal Clinical Trial Simulations

**Usage**

```
estSeqMarkovOrd(  
  y,  
  times,  
  initial,  
  absorb = NULL,  
  intercepts,  
  parameter,  
  looks,  
  g,  
  formula,  
  ppo = NULL,  
  yprevfactor = TRUE,  
  groupContrast = NULL,  
  cscov = FALSE,  
  timecriterion = NULL,  
  coxzph = FALSE,  
  sstat = NULL,  
  rdsample = NULL,  
  maxest = NULL,  
  maxvest = NULL,  
  nsim = 1,  
  progress = FALSE,  
  pfile = ""  
)
```

**Arguments**

y                    vector of possible y values in order (numeric, character, factor)  
times                vector of measurement times

<code>initial</code>	a vector of probabilities summing to 1.0 that specifies the frequency distribution of initial values to be sampled from. The vector must have names that correspond to values of <code>y</code> representing non-absorbing states.
<code>absorb</code>	vector of absorbing states, a subset of <code>y</code> . The default is no absorbing states. Observations are truncated when an absorbing state is simulated. May be numeric, character, or factor.
<code>intercepts</code>	vector of intercepts in the proportional odds model. There must be one fewer of these than the length of <code>y</code> .
<code>parameter</code>	vector of true parameter (effects; group differences) values. These are group 2:1 log odds ratios in the transition model, conditioning on the previous <code>y</code> .
<code>looks</code>	integer vector of ID numbers at which maximum likelihood estimates and their estimated variances are computed. For a single look specify a scalar value for loops equal to the number of subjects in the sample.
<code>g</code>	a user-specified function of three or more arguments which in order are <code>yprev</code> - the value of <code>y</code> at the previous time, the current time <code>t</code> , the gap between the previous time and the current time, an optional (usually named) covariate vector <code>X</code> , and optional arguments such as a regression coefficient value to simulate from. The function needs to allow <code>yprev</code> to be a vector and <code>yprev</code> must not include any absorbing states. The <code>g</code> function returns the linear predictor for the proportional odds model aside from <code>intercepts</code> . The returned value must be a matrix with row names taken from <code>yprev</code> . If the model is a proportional odds model, the returned value must be one column. If it is a partial proportional odds model, the value must have one column for each distinct value of the response variable <code>Y</code> after the first one, with the levels of <code>Y</code> used as optional column names. So columns correspond to <code>intercepts</code> . The different columns are used for <code>y</code> -specific contributions to the linear predictor (aside from <code>intercepts</code> ) for a partial or constrained partial proportional odds model. Parameters for partial proportional odds effects may be included in the ... arguments.
<code>formula</code>	a formula object given to the <code>lrm()</code> function using variables with these name: <code>y</code> , <code>time</code> , <code>yprev</code> , and <code>group</code> (factor variable having values '1' and '2'). The <code>yprev</code> variable is converted to a factor before fitting the model unless <code>yprevfactor=FALSE</code> .
<code>ppo</code>	a formula specifying the part of <code>formula</code> for which proportional odds is not to be assumed, i.e., that specifies a partial proportional odds model. Specifying <code>ppo</code> triggers the use of <code>VGAM::vglm()</code> instead of <code>rms::lrm</code> and will make the simulations run slower.
<code>yprevfactor</code>	see <code>formula</code>
<code>groupContrast</code>	omit this argument if <code>group</code> has only one regression coefficient in <code>formula</code> . Otherwise if <code>ppo</code> is omitted, provide <code>groupContrast</code> as a list of two lists that are passed to <code>rms::contrast.rms()</code> to compute the contrast of interest and its standard error. The first list corresponds to group 1, the second to group 2, to get a 2:1 contrast. If <code>ppo</code> is given and the group effect is not just a simple regression coefficient, specify as <code>groupContrast</code> a function of a <code>vglm</code> fit that computes the contrast of interest and its standard error and returns a list with elements named <code>Contrast</code> and <code>SE</code> . For the latter type you can optionally have formal arguments <code>n1</code> , <code>n2</code> , and <code>parameter</code> that are passed to <code>groupContrast</code> to compute the standard error of the group contrast, where <code>n1</code> and <code>n2</code> respectively

	are the sample sizes for the two groups and parameter is the true group effect parameter value.
cscov	applies if ppo is not used. Set to TRUE to use the cluster sandwich covariance estimator of the variance of the group comparison.
timecriterion	a function of a time-ordered vector of simulated ordinal responses y that returns a vector FALSE or TRUE values denoting whether the current y level met the condition of interest. For example estSeqMarkovOrd will compute the first time at which $y \geq 5$ if you specify timecriterion=function(y) y >= 5. This function is only called at the last data look for each simulated study. To have more control, instead of timecriterion returning a logical vector have it return a numeric 2-vector containing, in order, the event/censoring time and the 1/0 event/censoring indicator.
coxzph	set to TRUE if timecriterion is specified and you want to compute a statistic for testing proportional hazards at the last look of each simulated data
sstat	set to a function of the time vector and the corresponding vector of ordinal responses for a single group if you want to compute a Wilcoxon test on a derived quantity such as the number of days in a given state.
rdsample	an optional function to do response-dependent sampling. It is a function of these arguments, which are vectors that stop at any absorbing state: times (ascending measurement times for one subject), y (vector of ordinal outcomes at these times for one subject). The function returns NULL if no observations are to be dropped, returns the vector of new times to sample.
maxest	maximum acceptable absolute value of the contrast estimate, ignored if NULL. Any values exceeding maxest will result in the estimate being set to NA.
maxvest	like maxest but for the estimated variance of the contrast estimate
nsim	number of simulations (default is 1)
progress	set to TRUE to send current iteration number to pfile every 10 iterations. Each iteration will really involve multiple simulations, if parameter has length greater than 1.
pfile	file to which to write progress information. Defaults to ' ' which is the console. Ignored if progress=FALSE.

## Details

Simulates sequential clinical trials of longitudinal ordinal outcomes using a first-order Markov model. Looks are done sequentially after subject ID numbers given in the vector looks with the earliest possible look being after subject 2. At each look, a subject's repeated records are either all used or all ignored depending on the sequent ID number. For each true effect parameter value, simulation, and at each look, runs a function to compute the estimate of the parameter of interest along with its variance. For each simulation, data are first simulated for the last look, and these data are sequentially revealed for earlier looks. The user provides a function g that has extra arguments specifying the true effect of parameter the treatment group expecting treatments to be coded 1 and 2. parameter is usually on the scale of a regression coefficient, e.g., a log odds ratio. Fitting is done using the rms::lrm() function, unless non-proportional odds is allowed in which case VGAM::vglm() is used. If timecriterion is specified, the function also, for the last data look only, computes the first time at which the criterion is satisfied for the subject or use the event time

and event/censoring indicator computed by `timecriterion`. The Cox/logrank chi-square statistic for comparing groups on the derived time variable is saved. If `coxzph=TRUE`, the survival package correlation coefficient `rho` from the scaled partial residuals is also saved so that the user can later determine to what extent the Markov model resulted in the proportional hazards assumption being violated when analyzing on the time scale. `vglm` is accelerated by saving the first successful fit for the largest sample size and using its coefficients as starting value for further `vglm` fits for any sample size for the same setting of parameter.

### Value

a data frame with number of rows equal to the product of `nsim`, the length of `looks`, and the length of `parameter`, with variables `sim`, `parameter`, `look`, `est` (log odds ratio for group), and `vest` (the variance of the latter). If `timecriterion` is specified the data frame also contains `loghr` (Cox log hazard ratio for group), `lrchisq` (chi-square from Cox test for group), and if `coxph=TRUE`, `phchisq`, the chi-square for testing proportional hazards. The attribute `etimefreq` is also present if `timecriterion` is present, and it provides the frequency distribution of derived event times by group and censoring/event indicator. If `sstat` is given, the attribute `sstat` is also present, and it contains an array with dimensions corresponding to simulations, parameter values within simulations, `id`, and a two-column subarray with columns `group` and `y`, the latter being the summary measure computed by the `sstat` function. The returned data frame also has attribute `lrmcoef` which are the last-look logistic regression coefficient estimates over the `nsim` simulations and the parameter settings, and an attribute `failures` which is a data frame containing the variables `reason` and `frequency` cataloging the reasons for unsuccessful model fits.

### Author(s)

Frank Harrell

### See Also

`gbayesSeqSim()`, `simMarkovOrd()`, <https://hbiostat.org/R/Hmisc/markov/>

---

estSeqSim

*estSeqSim*

---

### Description

Simulate Comparisons For Use in Sequential Clinical Trial Simulations

### Usage

```
estSeqSim(parameter, looks, gendat, fitter, nsim = 1, progress = FALSE)
```

**Arguments**

parameter	vector of true parameter (effects; group differences) values
looks	integer vector of observation numbers at which posterior probabilities are computed
gendat	a function of three arguments: true parameter value (scalar), sample size for first group, sample size for second group
fitter	a function of two arguments: 0/1 group indicator vector and the dependent variable vector
nsim	number of simulations (default is 1)
progress	set to TRUE to send current iteration number to the console

**Details**

Simulates sequential clinical trials. Looks are done sequentially at observation numbers given in the vector `looks` with the earliest possible look being at observation 2. For each true effect parameter value, simulation, and at each look, runs a function to compute the estimate of the parameter of interest along with its variance. For each simulation, data are first simulated for the last look, and these data are sequentially revealed for earlier looks. The user provides a function `gendat` that given a true effect of parameter and the two sample sizes (for treatment groups 1 and 2) returns a list with vectors `y1` and `y2` containing simulated data. The user also provides a function `fitter` with arguments `x` (group indicator 0/1) and `y` (response variable) that returns a 2-vector containing the effect estimate and its variance. `parameter` is usually on the scale of a regression coefficient, e.g., a log odds ratio.

**Value**

a data frame with number of rows equal to the product of `nsim`, the length of `looks`, and the length of `parameter`.

**Author(s)**

Frank Harrell

**See Also**

`gbayesSeqSim()`, `simMarkovOrd()`, `estSeqMarkovOrd()`

**Examples**

```
if (requireNamespace("rms", quietly = TRUE)) {
  # Run 100 simulations, 5 looks, 2 true parameter values
  # Total simulation time: 2s
  lfit <- function(x, y) {
    f <- rms::lrm.fit(x, y)
    k <- length(coef(f))
    c(coef(f)[k], vcov(f)[k, k])
  }
  gdat <- function(beta, n1, n2) {
```

```

# Cell probabilities for a 7-category ordinal outcome for the control group
p <- c(2, 1, 2, 7, 8, 38, 42) / 100

# Compute cell probabilities for the treated group
p2 <- pomodm(p=p, odds.ratio=exp(beta))
y1 <- sample(1 : 7, n1, p, replace=TRUE)
y2 <- sample(1 : 7, n2, p2, replace=TRUE)
list(y1=y1, y2=y2)
}

set.seed(1)
est <- estSeqSim(c(0, log(0.7)), looks=c(50, 75, 95, 100, 200),
                gendat=gdat,
                fitter=lfit, nsim=100)
head(est)
}

```

---

event.chart

---

*Flexible Event Chart for Time-to-Event Data*


---

## Description

Creates an event chart on the current graphics device. Also, allows user to plot legend on plot area or on separate page. Contains features useful for plotting data with time-to-event outcomes Which arise in a variety of studies including randomized clinical trials and non-randomized cohort studies. This function can use as input a matrix or a data frame, although greater utility and ease of use will be seen with a data frame.

## Usage

```
event.chart(data, subset.r = 1:dim(data)[1], subset.c = 1:dim(data)[2],
```

```

  sort.by = NA, sort.ascending = TRUE,
  sort.na.last = TRUE, sort.after.subset = TRUE,
  y.var = NA, y.var.type = "n",
  y.jitter = FALSE, y.jitter.factor = 1,
  y.renum = FALSE, NA.rm = FALSE, x.reference = NA,
  now = max(data[, subset.c], na.rm = TRUE),
  now.line = FALSE, now.line.lty = 2,
  now.line.lwd = 1, now.line.col = 1, pty = "m",
  date.orig = c(1, 1, 1960), titl = "Event Chart",

```

```

  y.idlabels = NA, y.axis = "auto",
  y.axis.custom.at = NA, y.axis.custom.labels = NA,
  y.julian = FALSE, y.lim.extend = c(0, 0),
  y.lab = ifelse(is.na(y.idlabels), "", as.character(y.idlabels)),

```

```

  x.axis.all = TRUE, x.axis = "auto",

```



```

x.axis.custom.at = NA, x.axis.custom.labels = NA,
x.julian = FALSE, x.lim.extend = c(0, 0), x.scale = 1,
x.lab = ifelse(x.julian, "Follow-up Time", "Study Date"),

line.by = NA, line.lty = 1, line.lwd = 1, line.col = 1,
line.add = NA, line.add.lty = NA,
line.add.lwd = NA, line.add.col = NA,
point.pch = 1:length(subset.c),
point.cex = rep(0.6, length(subset.c)),
point.col = rep(1, length(subset.c)),

point.cex.mult = 1., point.cex.mult.var = NA,
extra.points.no.mult = rep(NA, length(subset.c)),

legend.plot = FALSE, legend.location = "o", legend.titl = titl,
legend.titl.cex = 3, legend.titl.line = 1,
legend.point.at = list(x = c(5, 95), y = c(95, 30)),
legend.point.pch = point.pch,
legend.point.text = ifelse(rep(is.data.frame(data), length(subset.c)),
                           names(data[, subset.c]),
                           subset.c),
legend.cex = 2.5, legend.bty = "n",
legend.line.at = list(x = c(5, 95), y = c(20, 5)),
legend.line.text = names(table(as.character(data[, line.by]),
                              exclude = c("", "NA"))),
legend.line.lwd = line.lwd, legend.loc.num = 1,

... )

```

### Arguments

<code>data</code>	a matrix or data frame with rows corresponding to subjects and columns corresponding to variables. Note that for a data frame or matrix containing multiple time-to-event data (e.g., time to recurrence, time to death, and time to last follow-up), one column is required for each specific event.
<code>subset.r</code>	subset of rows of original matrix or data frame to place in event chart. Logical arguments may be used here (e.g., <code>treatment.arm == 'a'</code> , if the data frame, <code>data</code> , has been attached to the search directory; otherwise, <code>data\$treatment.arm == "a"</code> ).
<code>subset.c</code>	subset of columns of original matrix or data frame to place in event chart; if working with a data frame, a vector of data frame variable names may be used for subsetting purposes (e.g., <code>c('randdate', 'event1')</code> ).
<code>sort.by</code>	column(s) or data frame variable name(s) with which to sort the chart's output. The default is <code>NA</code> , thereby resulting in a chart sorted by original row number.
<code>sort.ascending</code>	logical flag (which takes effect only if the argument <code>sort.by</code> is utilized). If <code>TRUE</code> (default), sorting is done in ascending order; if <code>FALSE</code> , descending order.
<code>sort.na.last</code>	logical flag (which takes effect only if the argument <code>sort.by</code> is utilized). If <code>TRUE</code> (default), <code>NA</code> values are considered as last values in ordering.

<code>sort.after.subset</code>	logical flag (which takes effect only if the argument <code>sort.by</code> is utilized). If <code>FALSE</code> , sorting data (via <code>sort.by</code> specified variables or columns) will be performed prior to row subsetting (via <code>subset.r</code> ); if <code>TRUE</code> (default), row subsetting of original data will be done before sorting.
<code>y.var</code>	variable name or column number of original matrix or data frame with which to scale y-axis. Default is <code>NA</code> , which will result in equally spaced lines on y-axis (based on original data or sorted data if requested by <code>sort.by</code> ). Otherwise, location of lines on y-axis will be dictated by specified variable or column. Examples of specified variables may be date of an event or a physiological covariate. Any observation which has a missing value for the <code>y.var</code> variable will not appear on the graph.
<code>y.var.type</code>	type of variable specified in <code>y.var</code> (which will only take effect if argument <code>y.var</code> is utilized). If <code>"d"</code> , specified variable is a date (either numeric julian date or an S-Plus dates object); if <code>"n"</code> , specified variable is numeric (e.g., systolic blood pressure level) although not a julian date.
<code>y.jitter</code>	logical flag (which takes effect only if the argument <code>y.var</code> is utilized). Due to potential ties in <code>y.var</code> variable, <code>y.jitter</code> (when <code>TRUE</code> ) will jitter the data to allow discrimination between observations at the possible cost of producing slightly inaccurate dates or covariate values; if <code>FALSE</code> (the default), no jittering will be performed. The <code>y.jitter</code> algorithm assumes a uniform distribution of observations across the range of <code>y.var</code> . The algorithm is as follows: <pre>size.jitter &lt;- ( diff(range(y.var)) / (2 * (length(y.var) - 1)) ) * y.jitter.factor</pre> <p>The default of <code>y.jitter.factor</code> is 1. The entire product is then used as an argument into <code>runif</code>: <code>y.var &lt;- y.var + runif(length(y.var), -size.jitter, size.jitter)</code></p>
<code>y.jitter.factor</code>	an argument used with the <code>y.jitter</code> function to scale the range of added noise. Default is 1.
<code>y.renum</code>	logical flag. If <code>TRUE</code> , subset observations are listed on y-axis from 1 to <code>length(subset.r)</code> ; if <code>FALSE</code> (default), subset observations are listed on y-axis in original form. As an example, if <code>subset.r = 301:340</code> and <code>y.renum == TRUE</code> , y-axis will be shown as 1 through 40. However, if <code>y.renum == FALSE</code> , y-axis will be shown as 301 through 340. The above examples assume the following argument, <code>NA.rm</code> , is set to <code>FALSE</code> .
<code>NA.rm</code>	logical flag. If <code>TRUE</code> , subset observations which have <code>NA</code> for each variable specified in <code>subset.c</code> will not have an entry on the y-axis. Also, if the following argument, <code>x.reference</code> , is specified, observations with missing <code>x.reference</code> values will also not have an entry on the y-axis. If <code>FALSE</code> (default), user can identify those observations which do have <code>NA</code> for every variable specified in <code>subset.c</code> (or, if <code>x.reference</code> is specified, also those observations which are missing only the <code>x.reference</code> value); this can easily be done by examining the resulting y-axis and recognizing the observations without any plotting symbols.
<code>x.reference</code>	column of original matrix or data frame with which to reference the x-axis. That is, if specified, all columns specified in <code>subset.c</code> will be subtracted by

	<code>x.reference</code> . An example may be to see the timing of events before and after treatment or to see time-to-event after entry into study. The event times will be aligned using the <code>x.reference</code> argument as the reference point.
<code>now</code>	the “now” date which will be used for top of y-axis when creating the Goldman eventchart (see reference below). Default is <code>max(data[, subset.c], na.rm=TRUE)</code> .
<code>now.line</code>	logical flag. A feature utilized by the Goldman Eventchart. When <code>x.reference</code> is specified as the start of follow-up and <code>y.var = x.reference</code> , then the Goldman chart can be created. This argument, if TRUE, will cause the plot region to be square, and will draw a line with a slope of -1 from the top of the y-axis to the right end of the x-axis. Essentially, it denotes end of current follow-up period for looking at the time-to-event data. Default is FALSE.
<code>now.line.lty</code>	line type of <code>now.line</code> .
<code>now.line.lwd</code>	line width of <code>now.line</code> .
<code>now.line.col</code>	color of <code>now.line</code> .
<code>pty</code>	graph option, <code>pty='m'</code> is the default; use <code>pty='s'</code> for the square looking Goldman’s event chart.
<code>date.orig</code>	date of origin to consider if dates are in julian, SAS , or S-Plus dates object format; default is January 1, 1960 (which is the default origin used by both S-Plus and SAS). Utilized when either <code>y.julian = FALSE</code> or <code>x.julian = FALSE</code> .
<code>titl</code>	title for event chart. Default is 'Event Chart'.
<code>y.idlabels</code>	column or data frame variable name used for y-axis labels. For example, if <code>c('pt.no')</code> is specified, patient ID (stored in <code>pt.no</code> ) will be seen on y-axis labels instead of sequence specified by <code>subset.r</code> . This argument takes precedence over both <code>y.axis = 'auto'</code> and <code>y.axis = 'custom'</code> (see below). NOTE: Program will issue warning if this argument is specified and if <code>is.na(y.var) == FALSE</code> ; <code>y.idlabels</code> will not be used in this situation. Also, attempting to plot too many patients on a single event chart will cause undesirable plotting of <code>y.idlabels</code> .
<code>y.axis</code>	character string specifying whether program will control labelling of y-axis (with argument "auto"), or if user will control labelling (with argument "custom"). If "custom" is chosen, user must specify location and text of labels using <code>y.axis.custom.at</code> and <code>y.axis.custom.labels</code> arguments, respectively, listed below. This argument will not be utilized if <code>y.idlabels</code> is specified.
<code>y.axis.custom.at</code>	user-specified vector of y-axis label locations. Must be used when <code>y.axis = "custom"</code> ; will not be used otherwise.
<code>y.axis.custom.labels</code>	user-specified vector of y-axis labels. Must be used when <code>y.axis = "custom"</code> ; will not be used otherwise.
<code>y.julian</code>	logical flag (which will only be considered if <code>y.axis == "auto"</code> and <code>(!is.na(y.var) &amp; y.var.type == "d")</code> ). If FALSE (default), will convert julian numeric dates or S-Plus dates objects into “mm/dd/yy” format for the y-axis labels. If TRUE, dates will be printed in julian (numeric) format.

<code>y.lim.extend</code>	two-dimensional vector representing the number of units that the user wants to increase <code>ylim</code> on bottom and top of y-axis, respectively. Default <code>c(0,0)</code> . This argument will not take effect if the Goldman chart is utilized.
<code>y.lab</code>	single label to be used for entire y-axis. Default will be the variable name or column number of <code>y.idlabels</code> (if non-missing) and blank otherwise.
<code>x.axis.all</code>	logical flag. If TRUE (default), lower and upper limits of x-axis will be based on all observations (rows) in matrix or data frame. If FALSE, lower and upper limits will be based only on those observations specified by <code>subset.r</code> (either before or after sorting depending on specification of <code>sort.by</code> and value of <code>sort.after.subset</code> ).
<code>x.axis</code>	character string specifying whether program will control labelling of x-axis (with argument "auto"), or if user will control labelling (with argument "custom"). If "custom" is chosen, user must specify location and text of labels using <code>x.axis.custom.at</code> and <code>x.axis.custom.labels</code> arguments, respectively, listed below.
<code>x.axis.custom.at</code>	user-specified vector of x-axis label locations. Must be used when <code>x.axis == "custom"</code> ; will not be used otherwise.
<code>x.axis.custom.labels</code>	user-specified vector of x-axis labels. Must be used when <code>x.axis == "custom"</code> ; will not be used otherwise.
<code>x.julian</code>	logical flag (which will only be considered if <code>x.axis == "auto"</code> ). If FALSE (default), will convert julian dates or S-plus dates objects into "mm/dd/yy" format for the x-axis labels. If TRUE, dates will be printed in julian (numeric) format. NOTE: This argument should remain TRUE if <code>x.reference</code> is specified.
<code>x.lim.extend</code>	two-dimensional vector representing the number of time units (usually in days) that the user wants to increase <code>xlim</code> on left-hand side and right-hand side of x-axis, respectively. Default is <code>c(0,0)</code> . This argument will not take effect if the Goldman chart is utilized.
<code>x.scale</code>	a factor whose reciprocal is multiplied to original units of the x-axis. For example, if the original data frame is in units of days, <code>x.scale = 365</code> will result in units of years (notwithstanding leap years). Default is 1.
<code>x.lab</code>	single label to be used for entire x-axis. Default will be "On Study Date" if <code>x.julian = FALSE</code> and "Time on Study" if <code>x.julian = TRUE</code> .
<code>line.by</code>	column or data frame variable name for plotting unique lines by unique values of vector (e.g., specify <code>c('arm')</code> to plot unique lines by treatment arm). Can take at most one column or variable name. Default is NA which produces identical lines for each patient.
<code>line.lty</code>	vector of line types corresponding to ascending order of <code>line.by</code> values. If <code>line.by</code> is specified, the vector should be the length of the number of unique values of <code>line.by</code> . If <code>line.by</code> is NA, only <code>line.lty[1]</code> will be used. The default is 1.
<code>line.lwd</code>	vector of line widths corresponding to ascending order of <code>line.by</code> values. If <code>line.by</code> is specified, the vector should be the length of the number of unique values of <code>line.by</code> . If <code>line.by</code> is NA, only <code>line.lwd[1]</code> will be used. The default is 1.

line.col	vector of line colors corresponding to ascending order of line.by values. If line.by is specified, the vector should be the length of the number of unique values of line.by. If line.by is NA, only line.col[1] will be used. The default is 1.
line.add	<p>a 2xk matrix with k=number of pairs of additional line segments to add. For example, if it is of interest to draw additional line segments connecting events one and two, two and three, and four and five, (possibly with different colors), an appropriate line.add argument would be matrix(c('first.event', 'second.event', 'second.event', 'fourth.event', 'fifth.event'), 2, 3). One line segment would be drawn between first.event and second.event, a second line segment would be drawn between second.event and third.event, and a third line segment would be drawn between fourth.event and fifth.event. Different line types, widths and colors can be specified (in arguments listed just below).</p> <p>The convention use of subset.c and line.add must match (i.e., column name must be used for both or column number must be used for both).</p> <p>If line.add != NA, length of line.add.lty, line.add.lwd, and line.add.col must be the same as number of pairs of additional line segments to add.</p> <p>NOTE: The drawing of the original default line may be suppressed (with line.col = 0), and line.add can be used to do all the line plotting for the event chart.</p>
line.add.lty	a kx1 vector corresponding to the columns of line.add; specifies the line types for the k line segments.
line.add.lwd	a kx1 vector corresponding to the columns of line.add; specifies the line widths for the k line segments.
line.add.col	a kx1 vector corresponding to the columns of line.add; specifies the line colors for the k line segments.
point.pch	vector of pch values for points representing each event. If similar events are listed in multiple columns (e.g., regular visits or a recurrent event), repeated pch values may be listed in the vector (e.g., c(2, 4, rep(183, 3))). If length(point.pch) < length(subset.c), point.pch will be repeated until lengths are equal; a warning message will verify this condition.
point.cex	vector of size of points representing each event. If length(point.cex) < length(subset.c), point.cex will be repeated until lengths are equal; a warning message will verify this condition.
point.col	vector of colors of points representing each event. If length(point.col) < length(subset.c), point.col will be repeated until lengths are equal; a warning message will verify this condition.
point.cex.mult	a single number (may be non-integer), which is the base multiplier for the value of the cex of the plotted points, when interest lies in a variable size allowed for certain points, as a function of the quantity of the variable(s) in the dataset specified in the point.cex.mult.var argument; multiplied by original point.cex value and then the value of interest (for an individual) from the point.cex.mult.var argument; used only when non-NA arguments are provided to point.cex.mult.var; default is 1. .
point.cex.mult.var	vector of variables to be used in determining what point.cex.mult is multiplied by for determining size of plotted points from (possibly a subset of) subset.c

variables, when interest lies in a variable size allowed for certain points, as a function of the level of some variable(s) in the dataset; default is NA.

<code>extra.points.no.mult</code>	vector of variables in the dataset to ignore for purposes of using <code>point.cex.mult</code> ; for example, for some variables there may be interest in allowing a variable size allowed for the plotting of the points, whereas other variables (e.g., dropout time), there may be no interest in such manipulation; the vector should be the same size as the number of variables specified in <code>subset.c</code> , with NA entries where variable point size is of interest and the variable name (or location in <code>subset.c</code> ) specified when the variable point size is not of interest; in this latter case, the associated argument in <code>point.cex</code> is instead used as the point <code>cex</code> ; used only when non-NA arguments are provided to <code>point.cex.mult.var</code> ; default is NA
<code>legend.plot</code>	logical flag; if TRUE, a legend will be plotted. Location of legend will be based on specification of <code>legend.location</code> along with values of other arguments listed below. Default is FALSE (i.e., no legend plotting).
<code>legend.location</code>	will be used only if <code>legend.plot = TRUE</code> . If "o" (default), a one-page legend will precede the output of the chart. The user will need to hit enter in order for the event chart to be displayed. This feature is possible due to the <code>dev.ask</code> option. If "i", an internal legend will be placed in the plot region based on <code>legend.point.at</code> . If "l", a legend will be placed in the plot region using the locator option. Legend will map points to events (via column names, by default) and, if <code>line.by</code> is specified, lines to groups (based on levels of <code>line.by</code> ).
<code>legend.titl</code>	title for the legend; default is title to be used for main plot. Only used when <code>legend.location = "o"</code> .
<code>legend.titl.cex</code>	size of text for legend title. Only used when <code>legend.location = "o"</code> .
<code>legend.titl.line</code>	line location of legend title dictated by <code>mtext</code> function with <code>outer = FALSE</code> option; default is 1.0. Only used when <code>legend.location = "o"</code> .
<code>legend.point.at</code>	location of upper left and lower right corners of legend area to be utilized for describing events via points and text.
<code>legend.point.pch</code>	vector of pch values for points representing each event in the legend. Default is <code>point.pch</code> .
<code>legend.point.text</code>	text to be used for describing events; the default is setup for a data frame, as it will print the names of the columns specified by <code>subset.c</code> .
<code>legend.cex</code>	size of text for points and event descriptions. Default is 2.5 which is setup for <code>legend.location = "o"</code> . A much smaller <code>cex</code> is recommended (possibly 0.75) for use with <code>legend.location = "i"</code> or <code>legend.location = "l"</code> .
<code>legend.bty</code>	option to put a box around the legend(s); default is to have no box ( <code>legend.bty = "n"</code> ). Option <code>legend.bty = "o"</code> will produce a legend box.

`legend.line.at` if `line.by` was specified (with `legend.location = "o"` or `legend.location = "i"`), this argument will dictate the location of the upper left and lower right corners of legend area to be utilized for describing the different `line.by` values (e.g., `treatment.arm`). The default is setup for `legend.location = "o"`.

`legend.line.text` text to be used for describing `line.by` values; the default are the names of the unique non-missing `line.by` values as produced from the `table` function.

`legend.line.lwd` vector of line widths corresponding to `line.by` values.

`legend.loc.num` number used for locator argument when `legend.locator = "1"`. If 1 (default), user is to locate only the top left corner of the legend box. If 2, user is to locate both the top left corner and the lower right corner. This will be done twice when `line.by` is specified (once for points and once for lines).

... additional `par` arguments for use in main plot.

### Details

if you want to put, say, two eventcharts side-by-side, in a plot region, you should not set up `par(mfrow=c(1,2))` before running the first plot. Instead, you should add the argument `mfg=c(1,1,1,2)` to the first plot call followed by the argument `mfg=c(1,2,1,2)` to the second plot call.

if dates in original data frame are in a specialized form (eg., mm/dd/yy) of mode CHARACTER, the user must convert those columns to become class dates or julian numeric mode (see [Date](#) for more information). For example, in a data frame called `testdata`, with specialized dates in columns 4 thru 10, the following code could be used: `as.numeric(dates(testdata[,4:10]))`. This will convert the columns to numeric julian dates based on the function's default origin of January 1, 1960. If original dates are in class dates or julian form, no extra work is necessary.

In the survival analysis, the data typically come in two columns: one column containing survival time and the other containing censoring indicator or event code. The `event.convert` function converts this type of data into multiple columns of event times, one column of each event type, suitable for the `event.chart` function.

### Side Effects

an event chart is created on the current graphics device. If `legend.plot = TRUE` and `legend.location = 'o'`, a one-page legend will precede the event chart. Please note that `par` parameters on completion of function will be reset to `par` parameters existing prior to start of function.

### Author(s)

J. Jack Lee and Kenneth R. Hess  
 Department of Biostatistics  
 University of Texas  
 M.D. Anderson Cancer Center  
 Houston, TX 77030  
[jjlee@mdanderson.org](mailto:jjlee@mdanderson.org), [khess@mdanderson.org](mailto:khess@mdanderson.org)  
 Joel A. Dubin  
 Department of Statistics

University of Waterloo  
<jdubin@uwaterloo.ca>

## References

- Lee J.J., Hess, K.R., Dubin, J.A. (2000). Extensions and applications of event charts. *The American Statistician*, **54:1**, 63–70.
- Dubin, J.A., Lee, J.J., Hess, K.R. (1997). The Utility of Event Charts. *Proceedings of the Biometrics Section, American Statistical Association*.
- Dubin, J.A., Muller H-G, Wang J-L (2001). Event history graphs for censored survival data. *Statistics in Medicine*, **20**: 2951–2964.
- Goldman, A.I. (1992). EVENTCHARTS: Visualizing Survival and Other Timed-Events Data. *The American Statistician*, **46:1**, 13–18.

## See Also

[event.history, Date](#)

## Examples

```
# The sample data set is an augmented CDC AIDS dataset (ASCII)
# which is used in the examples in the help file. This dataset is
# described in Kalbfleisch and Lawless (JASA, 1989).
# Here, we have included only children 4 years old and younger.
# We have also added a new field, dethdate, which
# represents a fictitious death date for each patient. There was
# no recording of death date on the original dataset. In addition, we have
# added a fictitious viral load reading (copies/ml) for each patient at time of AIDS diagnosis,
# noting viral load was also not part of the original dataset.
#
# All dates are julian with julian=0 being
# January 1, 1960, and julian=14000 being 14000 days beyond
# January 1, 1960 (i.e., May 1, 1998).

cdcaids <- data.frame(
age=c(4,2,1,1,2,2,2,4,2,1,1,3,2,1,3,2,1,2,4,2,2,1,4,2,4,1,4,2,1,1,3,3,1,3),
infedate=c(
7274,7727,7949,8037,7765,8096,8186,7520,8522,8609,8524,8213,8455,8739,
8034,8646,8886,8549,8068,8682,8612,9007,8461,8888,8096,9192,9107,9001,
9344,9155,8800,8519,9282,8673),
diagdate=c(
8100,8158,8251,8343,8463,8489,8554,8644,8713,8733,8854,8855,8863,8983,
9035,9037,9132,9164,9186,9221,9224,9252,9274,9404,9405,9433,9434,9470,
9470,9472,9489,9500,9585,9649),
diffdate=c(
826,431,302,306,698,393,368,1124,191,124,330,642,408,244,1001,391,246,
615,1118,539,612,245,813,516,1309,241,327,469,126,317,689,981,303,976),
dethdate=c(
8434,8304,NA,8414,8715,NA,8667,9142,8731,8750,8963,9120,9005,9028,9445,
9180,9189,9406,9711,9453,9465,9289,9640,9608,10010,9488,9523,9633,9667,
```



```

9547,9755,NA,9686,10084),
censdate=c(
NA,NA,8321,NA,NA,8519,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,
NA,NA,NA,NA,NA,NA,NA,NA,NA,10095,NA,NA),
viralload=c(
13000,36000,70000,90000,21000,110000,75000,12000,125000,110000,13000,39000,79000,135000,14000,
42000,123000,20000,12000,18000,16000,140000,16000,58000,11000,120000,85000,31000,24000,115000,
17000,13100,72000,13500)
)

```

```

cdcaids <- upData(cdcaids,
  labels=c(age      = 'Age, y', infedate='Date of blood transfusion',
           diagdate='Date of AIDS diagnosis',
           diffdate='Incubation period (days from HIV to AIDS)',
           dethdate='Fictitious date of death',
           censdate='Fictitious censoring date',
           viralload='Fictitious viral load'))

```

```

# Note that the style options listed with these
# examples are best suited for output to a postscript file (i.e., using
# the postscript function with horizontal=TRUE) as opposed to a graphical
# window (e.g., motif).

```

```

# To produce simple calendar event chart (with internal legend):
# postscript('example1.ps', horizontal=TRUE)
event.chart(cdcaids,
  subset.c=c('infedate','diagdate','dethdate','censdate'),
  x.lab = 'observation dates',
  y.lab='patients (sorted by AIDS diagnosis date)',
  titl='AIDS data calendar event chart 1',
  point.pch=c(1,2,15,0), point.cex=c(1,1,0.8,0.8),
  legend.plot=TRUE, legend.location='i', legend.cex=1.0,
  legend.point.text=c('transfusion','AIDS diagnosis','death','censored'),
  legend.point.at = list(c(7210, 8100), c(35, 27)), legend.bty='o')

```

```

# To produce simple interval event chart (with internal legend):
# postscript('example2.ps', horizontal=TRUE)
event.chart(cdcaids,
  subset.c=c('infedate','diagdate','dethdate','censdate'),
  x.lab = 'time since transfusion (in days)',
  y.lab='patients (sorted by AIDS diagnosis date)',
  titl='AIDS data interval event chart 1',
  point.pch=c(1,2,15,0), point.cex=c(1,1,0.8,0.8),
  legend.plot=TRUE, legend.location='i', legend.cex=1.0,
  legend.point.text=c('transfusion','AIDS diagnosis','death','censored'),
  x.reference='infedate', x.julian=TRUE,
  legend.bty='o', legend.point.at = list(c(1400, 1950), c(7, -1)))

```

```

# To produce simple interval event chart (with internal legend),

```

```

# but now with flexible diagdate symbol size based on viral load variable:
# postscript('example2a.ps', horizontal=TRUE)
event.chart(cdcaids,
  subset.c=c('infedate','diagdate','dethdate','censdate'),
  x.lab = 'time since transfusion (in days)',
  y.lab='patients (sorted by AIDS diagnosis date)',
  titl='AIDS data interval event chart 1a, with viral load at diagdate represented',
  point.pch=c(1,2,15,0), point.cex=c(1,1,0.8,0.8),
  point.cex.mult = 0.00002, point.cex.mult.var = 'viralload', extra.points.no.mult = c(1,NA,1,1),
  legend.plot=TRUE, legend.location='i', legend.cex=1.0,
  legend.point.text=c('transfusion','AIDS diagnosis','death','censored'),
  x.reference='infedate', x.julian=TRUE,
  legend.bty='o', legend.point.at = list(c(1400, 1950), c(7, -1)))

# To produce more complicated interval chart which is
# referenced by infection date, and sorted by age and incubation period:
# postscript('example3.ps', horizontal=TRUE)
event.chart(cdcaids,
  subset.c=c('infedate','diagdate','dethdate','censdate'),
  x.lab = 'time since diagnosis of AIDS (in days)',
  y.lab='patients (sorted by age and incubation length)',
  titl='AIDS data interval event chart 2 (sorted by age, incubation)',
  point.pch=c(1,2,15,0), point.cex=c(1,1,0.8,0.8),
  legend.plot=TRUE, legend.location='i', legend.cex=1.0,
  legend.point.text=c('transfusion','AIDS diagnosis','death','censored'),
  x.reference='diagdate', x.julian=TRUE, sort.by=c('age','diffdate'),
  line.by='age', line.lty=c(1,3,2,4), line.lwd=rep(1,4), line.col=rep(1,4),
  legend.bty='o', legend.point.at = list(c(-1350, -800), c(7, -1)),
  legend.line.at = list(c(-1350, -800), c(16, 8)),
  legend.line.text=c('age = 1', '          = 2', '          = 3', '          = 4'))

# To produce the Goldman chart:
# postscript('example4.ps', horizontal=TRUE)
event.chart(cdcaids,
  subset.c=c('infedate','diagdate','dethdate','censdate'),
  x.lab = 'time since transfusion (in days)', y.lab='dates of observation',
  titl='AIDS data Goldman event chart 1',
  y.var = c('infedate'), y.var.type='d', now.line=TRUE, y.jitter=FALSE,
  point.pch=c(1,2,15,0), point.cex=c(1,1,0.8,0.8), mgp = c(3.1,1.6,0),
  legend.plot=TRUE, legend.location='i', legend.cex=1.0,
  legend.point.text=c('transfusion','AIDS diagnosis','death','censored'),
  x.reference='infedate', x.julian=TRUE,
  legend.bty='o', legend.point.at = list(c(1500, 2800), c(9300, 10000)))

# To convert coded time-to-event data, then, draw an event chart:
surv.time <- c(5,6,3,1,2)
cens.ind  <- c(1,0,1,1,0)
surv.data <- cbind(surv.time,cens.ind)
event.data <- event.convert(surv.data)
event.chart(cbind(rep(0,5),event.data),x.julian=TRUE,x.reference=1)

```

---

`event.convert`*Event Conversion for Time-to-Event Data*

---

**Description**

Convert a two-column data matrix with event time and event code into multiple column event time with one event in each column

**Usage**

```
event.convert(data2, event.time = 1, event.code = 2)
```

**Arguments**

<code>data2</code>	a matrix or dataframe with at least 2 columns; by default, the first column contains the event time and the second column contains the k event codes (e.g. 1=dead, 0=censord)
<code>event.time</code>	the column number in data contains the event time
<code>event.code</code>	the column number in data contains the event code

**Details**

In the survival analysis, the data typically come in two columns: one column containing survival time and the other containing censoring indicator or event code. The `event.convert` function converts this type of data into multiple columns of event times, one column of each event type, suitable for the `event.chart` function.

**Author(s)**

J. Jack Lee and Kenneth R. Hess  
Department of Biostatistics  
University of Texas  
M.D. Anderson Cancer Center  
Houston, TX 77030  
<jjlee@mdanderson.org>, <khess@mdanderson.org>

Joel A. Dubin  
Department of Statistics  
University of Waterloo  
<jdubin@uwaterloo.ca>

**See Also**

[event.history](#), [Date](#), [event.chart](#)

## Examples

```
# To convert coded time-to-event data, then, draw an event chart:
surv.time <- c(5,6,3,1,2)
cens.ind   <- c(1,0,1,1,0)
surv.data  <- cbind(surv.time,cens.ind)
event.data <- event.convert(surv.data)
event.chart(cbind(rep(0,5),event.data),x.julian=TRUE,x.reference=1)
```

---

event.history

*Produces event.history graph for survival data*

---

## Description

Produces an event history graph for right-censored survival data, including time-dependent covariate status, as described in Dubin, Muller, and Wang (2001). Effectively, a Kaplan-Meier curve is produced with supplementary information regarding individual survival information, censoring information, and status over time of an individual time-dependent covariate or time-dependent covariate function for both uncensored and censored individuals.

## Usage

```
event.history(data, survtime.col, surv.col,
             surv.ind = c(1, 0), subset.rows = NULL,
             covtime.cols = NULL, cov.cols = NULL,
             num.colors = 1, cut.cov = NULL, colors = 1,
             cens.density = 10, mult.end.cens = 1.05,
             cens.mark.right = FALSE, cens.mark = "-",
             cens.mark.ahead = 0.5, cens.mark.cutoff = -1e-08,
             cens.mark.cex = 1,
             x.lab = "time under observation",
             y.lab = "estimated survival probability",
             title = "event history graph", ...)
```

## Arguments

data	A matrix or data frame with rows corresponding to units (often individuals) and columns corresponding to survival time, event/censoring indicator. Also, multiple columns may be devoted to time-dependent covariate level and time change.
survtime.col	Column (in data) representing minimum of time-to-event or right-censoring time for individual.
surv.col	Column (in data) representing event indicator for an individual. Though, traditionally, such an indicator will be 1 for an event and 0 for a censored observation, this indicator can be represented by any two numbers, made explicit by the surv.ind argument.

surv.ind	Two-element vector representing, respectively, the number for an event, as listed in <code>surv.col</code> , followed by the number for a censored observation. Default is traditional survival data representation, i.e., <code>c(1, 0)</code> .
subset.rows	Subset of rows of original matrix or data frame ( <code>data</code> ) to place in event history graph. Logical arguments may be used here (e.g., <code>treatment.arm == "a"</code> , if the data frame, <code>data</code> , has been attached to the search directory;
covtime.cols	Column(s) (in <code>data</code> ) representing the time when change of time-dependent covariate (or time-dependent covariate function) occurs. There should be a unique non-NA entry in the column for each such change (along with corresponding <code>cov.cols</code> column entry representing the value of the covariate or function at that change time). Default is NULL, meaning no time-dependent covariate information will be presented in the graph.
cov.cols	Column(s) (in <code>data</code> ) representing the level of the time-dependent covariate (or time-dependent covariate function). There should be a unique non-NA column entry representing each change in the level (along with a corresponding <code>covtime.cols</code> column entry representing the time of the change). Default is NULL, meaning no time-dependent covariate information will be presented in the graph.
num.colors	Colors are utilized for the time-dependent covariate level for an individual. This argument provides the number of unique covariate levels which will be displayed by mapping the number of colors (via <code>num.colors</code> ) to the number of desired covariate levels. This will divide the covariate span into roughly equally-sized intervals, via the S-Plus <code>cut</code> function. Default is one color, meaning no time-dependent information will be presented in the graph. Note that this argument will be ignored/superseded if a non-NULL argument is provided for the <code>cut.cov</code> parameter.
cut.cov	This argument allows the user to explicitly state how to define the intervals for the time-dependent covariate, such that different colors will be allocated to the user-defined covariate levels. For example, for plotting five colors, six ordered points within the span of the data's covariate levels should be provided. Default is NULL, meaning that the <code>num.colors</code> argument value will dictate the number of breakpoints, with the covariate span defined into roughly equally-sized intervals via the S-Plus <code>cut</code> function. However, if <code>is.null(cut.cov) == FALSE</code> , then this argument supercedes any entry for the <code>num.colors</code> argument.
colors	This is a vector argument defining the actual colors used for the time-dependent covariate levels in the plot, with the index of this vector corresponding to the ordered levels of the covariate. The number of colors (i.e., the length of the colors vector) should correspond to the value provided to the <code>num.colors</code> argument or the number of ordered points - 1 as defined in the <code>cut.cov</code> argument (with <code>cut.cov</code> superceding <code>num.colors</code> if <code>is.null(cut.cov) == FALSE</code> ). The function, as currently written, allows for as much as twenty distinct colors. This argument effectively feeds into the <code>col</code> argument for the S-Plus <code>polygon</code> function. Default is <code>colors = 1</code> . See the <code>col</code> argument for the both the S-Plus <code>par</code> function and <code>polygon</code> function for more information.
cens.density	This will provide the shading density at the end of the individual bars for those who are censored. For more information on shading density, see the <code>density</code> argument in the S-Plus <code>polygon</code> function. Default is <code>cens.density=10</code> .

<code>mult.end.cens</code>	This is a multiplier that extends the length of the longest surviving individual bar (or bars, if a tie exists) if right-censored, presuming that no event times eventually follow this final censored time. Default extends the length 5 percent beyond the length of the observed right-censored survival time.
<code>cens.mark.right</code>	A logical argument that states whether an explicit mark should be placed to the right of the individual right-censored survival bars. This argument is most useful for large sample sizes, where it may be hard to detect the special shading via <code>cens.density</code> , particularly for the short-term survivors.
<code>cens.mark</code>	Character argument which describes the censored mark that should be used if <code>cens.mark.right = TRUE</code> . Default is "-".
<code>cens.mark.ahead</code>	A numeric argument, which specifies the absolute distance to be placed between the individual right-censored survival bars and the mark as defined in the above <code>cens.mark</code> argument. Default is 0.5 (that is, a half of day, if survival time is measured in days), but may very well need adjusting depending on the maximum survival time observed in the dataset.
<code>cens.mark.cutoff</code>	A negative number very close to 0 (by default <code>cens.mark.cutoff = -1e-8</code> ) to ensure that the censoring marks get plotted correctly. See <code>event.history</code> code in order to see its usage. This argument typically will not need adjustment.
<code>cens.mark.cex</code>	Numeric argument defining the size of the mark defined in the <code>cens.mark</code> argument above. See more information by viewing the <code>cex</code> argument for the <code>S-Plus par</code> function. Default is <code>cens.mark.cex = 1.0</code> .
<code>x.lab</code>	Single label to be used for entire x-axis. Default is "time under observation".
<code>y.lab</code>	Single label to be used for entire y-axis. Default is "estimated survival probability".
<code>title</code>	Title for the event history graph. Default is "event history graph".
<code>...</code>	This allows arguments to the plot function call within the <code>event.history</code> function. So, for example, the axes representations can be manipulated with appropriate arguments, or particular areas of the <code>event.history</code> graph can be "zoomed". See the details section for more comments about zooming.

## Details

In order to focus on a particular area of the event history graph, zooming can be performed. This is best done by specifying appropriate `xlim` and `ylim` arguments at the end of the `event.history` function call, taking advantage of the `...` argument link to the plot function. An example of zooming can be seen in Plate 4 of the paper referenced below.

Please read the reference below to understand how the individual covariate and survival information is provided in the plot, how ties are handled, how right-censoring is handled, etc.

## WARNING

This function has been tested thoroughly, but only within a restricted version and environment, i.e., only within S-Plus 2000, Version 3, and within S-Plus 6.0, version 2, both on a Windows 2000

machine. Hence, we cannot currently vouch for the function's effectiveness in other versions of S-Plus (e.g., S-Plus 3.4) nor in other operating environments (e.g., Windows 95, Linux or Unix). The function has also been verified to work on R under Linux.

### Note

The authors have found better control of the use of color by producing the graphs via the postscript plotting device in S-Plus. In fact, the provided examples utilize the postscript function. However, your past experiences may be different, and you may prefer to control color directly (to the graphsheet in Windows environment, for example). The event.history function will work with either approach.

### Author(s)

Joel Dubin  
<jdubin@uwaterloo.ca>

### References

Dubin, J.A., Muller, H.-G., and Wang, J.-L. (2001). Event history graphs for censored survival data. *Statistics in Medicine*, **20**, 2951-2964.

### See Also

[plot,polygon](#), [event.chart](#), [par](#)

### Examples

```
# Code to produce event history graphs for SIM paper
#
# before generating plots, some pre-processing needs to be performed,
# in order to get dataset in proper form for event.history function;
# need to create one line per subject and sort by time under observation,
# with those experiencing event coming before those tied with censoring time;
require('survival')
data(heart)

# creation of event.history version of heart dataset (call heart.one):

heart.one <- matrix(nrow=length(unique(heart$id)), ncol=8)
for(i in 1:length(unique(heart$id)))
{
  if(length(heart$id[heart$id==i]) == 1)
    heart.one[i,] <- as.numeric(unlist(heart[heart$id==i, ]))
  else if(length(heart$id[heart$id==i]) == 2)
    heart.one[i,] <- as.numeric(unlist(heart[heart$id==i,][2,]))
}

heart.one[,3][heart.one[,3] == 0] <- 2 ## converting censored events to 2, from 0
if(is.factor(heart$transplant))
  heart.one[,7] <- heart.one[,7] - 1
## getting back to correct transplantation coding
```

```

heart.one <- as.data.frame(heart.one[order(unlist(heart.one[,2]), unlist(heart.one[,3])),])
names(heart.one) <- names(heart)
# back to usual censoring indicator:
heart.one[,3][heart.one[,3] == 2] <- 0
# note: transplant says 0 (for no transplants) or 1 (for one transplant)
#       and event = 1 is death, while event = 0 is censored

# plot single Kaplan-Meier curve from heart data, first creating survival object
heart.surv <- survfit(Surv(stop, event) ~ 1, data=heart.one, conf.int = FALSE)

# figure 3: traditional Kaplan-Meier curve
# postscript('ehgfig3.ps', horiz=TRUE)
# omi <- par(omi=c(0,1.25,0.5,1.25))
plot(heart.surv, ylab='estimated survival probability',
      xlab='observation time (in days)')
title('Figure 3: Kaplan-Meier curve for Stanford data', cex=0.8)
# dev.off()

## now, draw event history graph for Stanford heart data; use as Figure 4

# postscript('ehgfig4.ps', horiz=TRUE, colors = seq(0, 1, len=20))
# par(omi=c(0,1.25,0.5,1.25))
event.history(heart.one,
survtime.col=heart.one[,2], surv.col=heart.one[,3],
covtime.cols = cbind(rep(0, dim(heart.one)[1]), heart.one[,1]),
cov.cols = cbind(rep(0, dim(heart.one)[1]), heart.one[,7]),
num.colors=2, colors=c(6,10),
x.lab = 'time under observation (in days)',
title='Figure 4: Event history graph for\nStanford data',
cens.mark.right =TRUE, cens.mark = '-',
cens.mark.ahead = 30.0, cens.mark.cex = 0.85)
# dev.off()

# now, draw age-stratified event history graph for Stanford heart data;
# use as Figure 5

# two plots, stratified by age status
# postscript('c:\temp\ehgfig5.ps', horiz=TRUE, colors = seq(0, 1, len=20))
# par(omi=c(0,1.25,0.5,1.25))
par(mfrow=c(1,2))

event.history(data=heart.one, subset.rows = (heart.one[,4] < 0),
survtime.col=heart.one[,2], surv.col=heart.one[,3],
covtime.cols = cbind(rep(0, dim(heart.one)[1]), heart.one[,1]),
cov.cols = cbind(rep(0, dim(heart.one)[1]), heart.one[,7]),
num.colors=2, colors=c(6,10),
x.lab = 'time under observation\n(in days)',
title = 'Figure 5a:\nStanford data\n(age < 48)',
cens.mark.right =TRUE, cens.mark = '-',
cens.mark.ahead = 40.0, cens.mark.cex = 0.85,
xlim=c(0,1900))

```



```

event.history(data=heart.one, subset.rows = (heart.one[,4] >= 0),
survtime.col=heart.one[,2], surv.col=heart.one[,3],
covtime.cols = cbind(rep(0, dim(heart.one)[1]), heart.one[,1]),
cov.cols = cbind(rep(0, dim(heart.one)[1]), heart.one[,7]),
num.colors=2, colors=c(6,10),
x.lab = 'time under observation\n(in days)',
title = 'Figure 5b:\nStanford data\n(age >= 48)',
cens.mark.right =TRUE, cens.mark = '-',
cens.mark.ahead = 40.0, cens.mark.cex = 0.85,
xlim=c(0,1900))
# dev.off()
# par(omi=omi)

# we will not show liver cirrhosis data manipulation, as it was
# a bit detailed; however, here is the
# event.history code to produce Figure 7 / Plate 1

# Figure 7 / Plate 1 : prothrombin ehg with color
## Not run:
second.arg <- 1 ### second.arg is for shading
third.arg <- c(rep(1,18),0,1) ### third.arg is for intensity

# postscript('c:\temp\ehgfig7.ps', horiz=TRUE,
# colors = cbind(seq(0, 1, len = 20), second.arg, third.arg))
# par(omi=c(0,1.25,0.5,1.25), col=19)
event.history(cirrhos2.eh, subset.rows = NULL,
survtime.col=cirrhos2.eh$time, surv.col=cirrhos2.eh$event,
covtime.cols = as.matrix(cirrhos2.eh[, ((2:18)*2)]),
cov.cols = as.matrix(cirrhos2.eh[, ((2:18)*2) + 1]),
cut.cov = as.numeric(quantile(as.matrix(cirrhos2.eh[, ((2:18)*2) + 1]),
c(0,.2,.4,.6,.8,1), na.rm=TRUE) + c(-1,0,0,0,0,1)),
colors=c(20,4,8,11,14),
x.lab = 'time under observation (in days)',
title='Figure 7: Event history graph for liver cirrhosis data (color)',
cens.mark.right =TRUE, cens.mark = '-',
cens.mark.ahead = 100.0, cens.mark.cex = 0.85)
# dev.off()

## End(Not run)

```

---

extractlabs

*extractlabs*


---

## Description

Extract Labels and Units From Multiple Datasets

## Usage

```
extractlabs(..., print = TRUE)
```

**Arguments**

... one ore more data frames or data tables  
print set to FALSE to not print details about variables with conflicting attributes

**Details**

For one or more data frames/tables extracts all labels and units and comb ines them over dataset, dropping any variables not having either labels or units defined. The resulting data table is returned and is used by the hlab function if the user stores the result in an objectnamed LabelsUnits. The result is NULL if no variable in any dataset has a non-blank label or units. Variables found in more than one dataset with duplicate label and units are consolidated. A warning message is issued when duplicate variables have conflicting labels or units, and by default, details are printed. No attempt is made to resolve these conflicts.

**Value**

a data table

**Author(s)**

Frank Harrell

**See Also**

[label\(\)](#), [contents\(\)](#), [units\(\)](#), [hlab\(\)](#)

**Examples**

```
d <- data.frame(x=1:10, y=(1:10)/10)
d <- upData(d, labels=c(x='X', y='Y'), units=c(x='mmHg'), print=FALSE)
d2 <- d
units(d2$x) <- 'cm'
LabelsUnits <- extractlabs(d, d2)
LabelsUnits
```

---

fImport

*fImport*

---

**Description**

General File Import Using rio

**Usage**

```
fImport(
  file,
  format,
  lowernames = c("not mixed", "no", "yes"),
  und. = FALSE,
  ...
)
```

**Arguments**

file	name of file to import, or full URL. rio determines the file type from the file suffix unless you override this with format
format	format of file to import, usually not needed. See <code>rio::import()</code> for details
lowernames	defaults to changing variable names to all lower case unless the name as mixed upper and lower case, which results in keeping the original characters in the name. Set <code>lowernames='no'</code> to leave variable names as they were created in the original file export, or set <code>lowernames='yes'</code> to set all names to lower case whether they have mixed case or not. For all options, a check is made to see if the name conversions would result in any duplicate names. If so, the original names are retained and a warning message issued.
und.	set to TRUE to change all underscores in names to periods
...	more arguments to pass to <code>rio::import()</code>

**Details**

This is a front-end for the `rio` package's `import` function. `fImport` includes options for setting variable names to lower case and to change underscores in names to periods. Variables on the imported data frame that have labels are converted to `Hmisc` package labelled class so that subsetting the data frame will preserve the labels.

**Value**

a data frame created by `rio`, unless a `rio` option is given to use another format

**Author(s)**

Frank Harrell

**See Also**

`upData`, especially the `moveUnits` option

**Examples**

```
## Not run:
# Get a Stata dataset
d <- fImport('http://www.principlesofeconometrics.com/stata/beer.dta')
```

```

contents(d)

## End(Not run)

```

---

```
find.matches
```

```
Find Close Matches
```

---

## Description

Compares each row in *x* against all the rows in *y*, finding rows in *y* with all columns within a tolerance of the values a given row of *x*. The default tolerance *tol* is zero, i.e., an exact match is required on all columns. For qualifying matches, a distance measure is computed. This is the sum of squares of differences between *x* and *y* after scaling the columns. The default scaling values are *tol*, and for columns with *tol*=1 the scale values are set to 1.0 (since they are ignored anyway). Matches (up to *maxmatch* of them) are stored and listed in order of increasing distance.

The summary method prints a frequency distribution of the number of matches per observation in *x*, the median of the minimum distances for all matches per *x*, as a function of the number of matches, and the frequency of selection of duplicate observations as those having the smallest distance. The print method prints the entire matches and distance components of the result from *find.matches*.

*matchCases* finds all controls that match cases on a single variable *x* within a tolerance of *tol*. This is intended for prospective cohort studies that use matching for confounder adjustment (even though regression models usually work better).

## Usage

```

find.matches(x, y, tol=rep(0, ncol(y)), scale=tol, maxmatch=10)
## S3 method for class 'find.matches'
summary(object, ...)
## S3 method for class 'find.matches'
print(x, digits, ...)

matchCases(xcase, ycase, idcase=names(ycase),
           xcontrol, ycontrol, idcontrol=names(ycontrol),
           tol=NULL,
           maxobs=max(length(ycase),length(ycontrol))*10,
           maxmatch=20, which=c('closest','random'))

```

## Arguments

<i>x</i>	a numeric matrix or the result of <i>find.matches</i>
<i>y</i>	a numeric matrix with same number of columns as <i>x</i>
<i>xcase</i>	numeric vector to match on for cases
<i>xcontrol</i>	numeric vector to match on for controls, not necessarily the same length as <i>xcase</i>
<i>ycase</i>	a vector or matrix

ycontrol	ycase and ycontrol are vectors or matrices, not necessarily having the same number of rows, specifying a variable to carry along from cases and matching controls. If you instead want to carry along rows from a data frame, let ycase and ycontrol be non-overlapping integer subscripts of the donor data frame.
tol	a vector of tolerances with number of elements the same as the number of columns of y, for find.matches. For matchCases is a scalar tolerance.
scale	a vector of scaling constants with number of elements the same as the number of columns of y.
maxmatch	maximum number of matches to allow. For matchCases, maximum number of controls to match with a case (default is 20). If more than maxmatch matching controls are available, a random sample without replacement of maxmatch controls is used (if which="random").
object	an object created by find.matches
digits	number of digits to use in printing distances
idcase	vector the same length as xcase
idcontrol	idcase and idcontrol are vectors the same length as xcase and xcontrol respectively, specifying the id of cases and controls. Defaults are integers specifying original element positions within each of cases and controls.
maxobs	maximum number of cases and all matching controls combined (maximum dimension of data frame resulting from matchControls). Default is ten times the maximum of the number of cases and number of controls. maxobs is used to allocate space for the resulting data frame.
which	set to "closest" (the default) to match cases with up to maxmatch controls that most closely match on x. Set which="random" to use randomly chosen controls. In either case, only those controls within tol on x are allowed to be used.
...	unused

### Value

find.matches returns a list of class find.matches with elements matches and distance. Both elements are matrices with the number of rows equal to the number of rows in x, and with k columns, where k is the maximum number of matches ( $\leq$  maxmatch) that occurred. The elements of matches are row identifiers of y that match, with zeros if fewer than maxmatch matches are found (blanks if y had row names). matchCases returns a data frame with variables idcase (id of case currently being matched), type (factor variable with levels "case" and "control"), id (id of case if case row, or id of matching case), and y.

### Author(s)

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>

## References

Ming K, Rosenbaum PR (2001): A note on optimal matching with variable controls using the assignment algorithm. *J Comp Graph Stat* 10:455–463.

Cepeda MS, Boston R, Farrar JT, Strom BL (2003): Optimal matching with a variable number of controls vs. a fixed number of controls for a cohort study: trade-offs. *J Clin Epidemiology* 56:230-237. Note: These papers were not used for the functions here but probably should have been.

## See Also

[scale](#), [apply](#)

## Examples

```

y <- rbind(c(.1, .2),c(.11, .22), c(.3, .4), c(.31, .41), c(.32, 5))
x <- rbind(c(.09,.21), c(.29,.39))
y
x
w <- find.matches(x, y, maxmatch=5, tol=c(.05,.05))

set.seed(111)      # so can replicate results
x <- matrix(runif(500), ncol=2)
y <- matrix(runif(2000), ncol=2)
w <- find.matches(x, y, maxmatch=5, tol=c(.02,.03))
w$matches[1:5,]
w$distance[1:5,]
# Find first x with 3 or more y-matches
num.match <- apply(w$matches, 1, function(x)sum(x > 0))
j <- ((1:length(num.match))[num.match > 2])[1]
x[j,]
y[w$matches[j,],]

summary(w)

# For many applications would do something like this:
# attach(df1)
# x <- cbind(age, sex) # Just do as.matrix(df1) if df1 has no factor objects
# attach(df2)
# y <- cbind(age, sex)
# mat <- find.matches(x, y, tol=c(5,0)) # exact match on sex, 5y on age

# Demonstrate matchCases
xcase <- c(1,3,5,12)
xcontrol <- 1:6
idcase <- c('A','B','C','D')
idcontrol <- c('a','b','c','d','e','f')
ycase <- c(11,33,55,122)
ycontrol <- c(11,22,33,44,55,66)

```

```

matchCases(xcase, ycase, idcase,
           xcontrol, ycontrol, idcontrol, tol=1)

# If y is a binary response variable, the following code
# will produce a Mantel-Haenszel summary odds ratio that
# utilizes the matching.
# Standard variance formula will not work here because
# a control will match more than one case
# WARNING: The M-H procedure exemplified here is suspect
# because of the small strata and widely varying number
# of controls per case.

x  <- c(1, 2, 3, 3, 3, 6, 7, 12, 1, 1:7)
y  <- c(0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1)
case <- c(rep(TRUE, 8), rep(FALSE, 8))
id  <- 1:length(x)

m <- matchCases(x[case], y[case], id[case],
                x[!case], y[!case], id[!case], tol=1)
iscase <- m$type=='case'
# Note: the first tapply on insures that event indicators are
# sorted by case id. The second actually does something.
event.case   <- tapply(m$y[iscase], m$idcase[iscase], sum)
event.control <- tapply(m$y[!iscase], m$idcase[!iscase], sum)
n.control    <- tapply(!iscase, m$idcase, sum)
n            <- tapply(m$y, m$idcase, length)
or <- sum(event.case * (n.control - event.control) / n) /
      sum(event.control * (1 - event.case) / n)
or

# Bootstrap this estimator by sampling with replacement from
# subjects. Assumes id is unique when combine cases+controls
# (id was constructed this way above). The following algorithms
# puts all sampled controls back with the cases to whom they were
# originally matched.

ids <- unique(m$id)
idgroups <- split(1:nrow(m), m$id)
B <- 50 # in practice use many more
ors <- numeric(B)
# Function to order w by ids, leaving unassigned elements zero
align <- function(ids, w) {
  z <- structure(rep(0, length(ids)), names=ids)
  z[names(w)] <- w
  z
}
for(i in 1:B) {
  j <- sample(ids, replace=TRUE)

```

```

obs <- unlist(idgroups[j])
u <- m[obs,]
iscase <- u$type=='case'
n.case <- align(ids, tapply(u$type, u$idcase,
                           function(v)sum(v=='case')))
n.control <- align(ids, tapply(u$type, u$idcase,
                              function(v)sum(v=='control')))
event.case <- align(ids, tapply(u$y[iscase], u$idcase[iscase], sum))
event.control <- align(ids, tapply(u$y[!iscase], u$idcase[!iscase], sum))
n <- n.case + n.control
# Remove sets having 0 cases or 0 controls in resample
s <- n.case > 0 & n.control > 0
denom <- sum(event.control[s] * (n.case[s] - event.case[s]) / n[s])
or <- if(denom==0) NA else
  sum(event.case[s] * (n.control[s] - event.control[s]) / n[s]) / denom
ors[i] <- or
}
describe(ors)

```

---

first.word

*First Word in a String or Expression*


---

### Description

`first.word` finds the first word in an expression. A word is defined by unlisting the elements of the expression found by the S parser and then accepting any elements whose first character is either a letter or period. The principal intended use is for the automatic generation of temporary file names where it is important to exclude special characters from the file name. For Microsoft Windows, periods in names are deleted and only up to the first 8 characters of the word is returned.

### Usage

```
first.word(x, i=1, expr=substitute(x))
```

### Arguments

<code>x</code>	any scalar character string
<code>i</code>	word number, default value = 1. Used when the second or <i>i</i> th word is wanted. Currently only the <i>i</i> =1 case is implemented.
<code>expr</code>	any S object of mode expression.

### Value

a character string



**Author(s)**

Frank E. Harrell, Jr.,  
 Department of Biostatistics,  
 Vanderbilt University,  
 <fh@fharrell.com>

Richard M. Heiberger,  
 Department of Statistics,  
 Temple University, Philadelphia, PA.  
 <rmh@temple.edu>

**Examples**

```
first.word(expr=expression(y ~ x + log(w)))
```

---

 format.df

---

*Format a Data Frame or Matrix for LaTeX or HTML*


---

**Description**

format.df does appropriate rounding and decimal alignment, and outputs a character matrix containing the formatted data. If `x` is a `data.frame`, then do each component separately. If `x` is a matrix, but not a `data.frame`, make it a `data.frame` with individual components for the columns. If a component `x$x` is a matrix, then do all columns the same.

**Usage**

```
format.df(x, digits, dec=NULL, rdec=NULL, cdec=NULL,
  numeric.dollar=!dcolumn, na.blank=FALSE, na.dot=FALSE,
  blank.dot=FALSE, col.just=NULL, cdot=FALSE,
  dcolumn=FALSE, matrix.sep=' ', scientific=c(-4,4),
  math.row.names=FALSE, already.math.row.names=FALSE,
  math.col.names=FALSE, already.math.col.names=FALSE,
  double.slash=FALSE, format.Date="%m/%d/%Y",
  format.POSIXt="%m/%d/%Y %H:%M:%OS", ...)
```

**Arguments**

<code>x</code>	a matrix (usually numeric) or data frame
<code>digits</code>	causes all values in the table to be formatted to <code>digits</code> significant digits. <code>dec</code> is usually preferred.
<code>dec</code>	If <code>dec</code> is a scalar, all elements of the matrix will be rounded to <code>dec</code> decimal places to the right of the decimal. <code>dec</code> can also be a matrix whose elements correspond to <code>x</code> , for customized rounding of each element. A matrix <code>dec</code> must have number of columns equal to number of columns of input <code>x</code> . A scalar <code>dec</code> is expanded to a vector <code>cdec</code> with number of items equal to number of columns of input <code>x</code> .

rdec	a vector specifying the number of decimal places to the right for each row (cdec is more commonly used than rdec) A vector rdec must have number of items equal to number of rows of input x. rdec is expanded to matrix dec.
cdec	a vector specifying the number of decimal places for each column. The vector must have number of items equal to number of columns or components of input x.
cdot	Set to TRUE to use centered dots rather than ordinary periods in numbers. The output uses a syntax appropriate for latex.
na.blank	Set to TRUE to use blanks rather than NA for missing values. This usually looks better in latex.
dcolumn	Set to TRUE to use David Carlisle's dcolumn style for decimal alignment in latex. Default is FALSE. You will probably want to use dcolumn if you use rdec, as a column may then contain varying number of places to the right of the decimal. dcolumn can line up all such numbers on the decimal point, with integer values right justified at the decimal point location of numbers that actually contain decimal places. When you use dcolumn = TRUE, numeric.dollar is set by default to FALSE. When you use dcolumn = TRUE, the object attribute "style" set to 'dcolumn' as the latex usepackage must reference [dcolumn]. The three files 'dcolumn.sty', 'newarray.sty', and 'array.sty' will need to be in a directory in your TEXINPUTS path. When you use dcolumn=TRUE, numeric.dollar should be set to FALSE.
numeric.dollar	logical, default !dcolumn. Set to TRUE to place dollar signs around numeric values when dcolumn = FALSE. This assures that latex will use minus signs rather than hyphens to indicate negative numbers. Set to FALSE when dcolumn = TRUE, as dcolumn.sty automatically uses minus signs.
math.row.names	logical, set true to place dollar signs around the row names.
already.math.row.names	set to TRUE to prevent any math mode changes to row names
math.col.names	logical, set true to place dollar signs around the column names.
already.math.col.names	set to TRUE to prevent any math mode changes to column names
na.dot	Set to TRUE to use periods rather than NA for missing numeric values. This works with the SAS convention that periods indicate missing values.
blank.dot	Set to TRUE to use periods rather than blanks for missing character values. This works with the SAS convention that periods indicate missing values.
col.just	Input vector col.just must have number of columns equal to number of columns of the output matrix. When NULL, the default, the col.just attribute of the result is set to 'l' for character columns and to 'r' for numeric columns. The user can override the default by an argument vector whose length is equal to the number of columns of the result matrix. When format.df is called by latex.default, the col.just is used as the cols argument to the tabular environment and the letters 'l', 'r', and 'c' are valid values. When format.df is called by SAS, the col.just is used to determine whether a '\\$' is needed on the 'input' line of the 'sysin' file, and the letters 'l' and 'r' are valid values. You can pass specifications other than l,r,c in col.just, e.g., "p{3in}" to get paragraph-formatted columns from latex().

<code>matrix.sep</code>	When <code>x</code> is a data frame containing a matrix, so that new column names are constructed from the name of the matrix object and the names of the individual columns of the matrix, <code>matrix.sep</code> specifies the character to use to separate object names from individual column names.
<code>scientific</code>	specifies ranges of exponents (or a logical vector) specifying values not to convert to scientific notation. See <code>format.default</code> for details.
<code>double.slash</code>	should escaping backslashes be themselves escaped.
<code>format.Date</code>	String used to format objects of the Date class.
<code>format.POSIXt</code>	String used to format objects of the POSIXt class.
<code>...</code>	other arguments are accepted and passed to <code>format.default</code> . For <code>latexVerbatim</code> these arguments are passed to the <code>print</code> function.

**Value**

a character matrix with character images of properly rounded `x`. Matrix components of input `x` are now just sets of columns of character matrix. Object attribute "`col.just`" repeats the value of the argument `col.just` when provided, otherwise, it includes the recommended justification for columns of output. See the discussion of the argument `col.just`. The default justification is 'l' for characters and factors, 'r' for numeric. When `dcolumn==TRUE`, numerics will have '.' as the justification character.

**Author(s)**

Frank E. Harrell, Jr.,  
 Department of Biostatistics,  
 Vanderbilt University,  
 <fh@fharrell.com>

Richard M. Heiberger,  
 Department of Statistics,  
 Temple University, Philadelphia, PA.  
 <rmh@temple.edu>

**See Also**

[latex](#)

**Examples**

```
## Not run:
x <- data.frame(a=1:2, b=3:4)
x$m <- 10000*matrix(5:8,nrow=2)
names(x)
dim(x)
x
format.df(x, big.mark=",")
dim(format.df(x))

## End(Not run)
```

---

`format.pval`*Format P Values*

---

### Description

`format.pval` is intended for formatting p-values.

### Usage

```
format.pval(x, pv=x, digits = max(1, .Options$digits - 2),  
           eps = .Machine$double.eps, na.form = "NA", ...)
```

### Arguments

<code>pv</code>	a numeric vector.
<code>x</code>	argument for method compliance.
<code>digits</code>	how many significant digits are to be used.
<code>eps</code>	a numerical tolerance: see Details.
<code>na.form</code>	character representation of NAs.
<code>...</code>	arguments passed to <code>format</code> in the <code>format.pval</code> function body.

### Details

`format.pval` is mainly an auxiliary function for `print.summary.lm` etc., and does separate formatting for fixed, floating point and very small values; those less than `eps` are formatted as “<[eps]” (where “<[eps]” stands for `format(eps, digits)`).

### Value

A character vector.

### Note

This is the base `format.pval` function with the ability to pass the `nsmall` argument to `format`

### Examples

```
format.pval(c(runif(5), pi^-100, NA))  
format.pval(c(0.1, 0.0001, 1e-27))  
format.pval(c(0.1, 1e-27), nsmall=3)
```

**Description**

gbayes derives the (Gaussian) posterior and optionally the predictive distribution when both the prior and the likelihood are Gaussian, and when the statistic of interest comes from a 2-sample problem. This function is especially useful in obtaining the expected power of a statistical test, averaging over the distribution of the population effect parameter (e.g., log hazard ratio) that is obtained using pilot data. gbayes is also useful for summarizing studies for which the statistic of interest is approximately Gaussian with known variance. An example is given for comparing two proportions using the angular transformation, for which the variance is independent of unknown parameters except for very extreme probabilities. A plot method is also given. This plots the prior, posterior, and predictive distributions on a single graph using a nice default for the x-axis limits and using the labcurve function for automatic labeling of the curves.

gbayes2 uses the method of Spiegelhalter and Freedman (1986) to compute the probability of correctly concluding that a new treatment is superior to a control. By this we mean that a  $1-\alpha$  normal theory-based confidence interval for the new minus old treatment effect lies wholly to the right of  $\delta.w$ , where  $\delta.w$  is the minimally worthwhile treatment effect (which can be zero to be consistent with ordinary null hypothesis testing, a method not always making sense). This kind of power function is averaged over a prior distribution for the unknown treatment effect. This procedure is applicable to the situation where a prior distribution is not to be used in constructing the test statistic or confidence interval, but is only used for specifying the distribution of  $\delta$ , the parameter of interest.

Even though gbayes2 assumes that the test statistic has a normal distribution with known variance (which is strongly a function of the sample size in the two treatment groups), the prior distribution function can be completely general. Instead of using a step-function for the prior distribution as Spiegelhalter and Freedman used in their appendix, gbayes2 uses the built-in integrate function for numerical integration. gbayes2 also allows the variance of the test statistic to be general as long as it is evaluated by the user. The conditional power given the parameter of interest  $\delta$  is  $1 - \text{pnorm}((\delta.w - \delta)/sd + z)$ , where  $z$  is the normal critical value corresponding to  $1 - \alpha/2$ .

gbayesMixPredNoData derives the predictive distribution of a statistic that is Gaussian given  $\delta$  when no data have yet been observed and when the prior is a mixture of two Gaussians.

gbayesMixPost derives the posterior density, cdf, or posterior mean of  $\delta$  given the statistic  $x$ , when the prior for  $\delta$  is a mixture of two Gaussians and when  $x$  is Gaussian given  $\delta$ .

gbayesMixPowerNP computes the power for a test for  $\delta > \delta.w$  for the case where (1) a Gaussian prior or mixture of two Gaussian priors is used as the prior distribution, (2) this prior is used in forming the statistical test or credible interval, (3) no prior is used for the distribution of  $\delta$  for computing power but instead a fixed single  $\delta$  is given (as in traditional frequentist hypothesis tests), and (4) the test statistic has a Gaussian likelihood with known variance (and mean equal to the specified  $\delta$ ). gbayesMixPowerNP is handy where you want to use an earlier study in testing for treatment effects in a new study, but you want to mix with this prior a non-informative prior. The mixing probability  $\text{mix}$  can be thought of as the "applicability" of the previous study. As with gbayes2, power here means the probability that the new study will yield a left credible interval

that is to the right of  $\delta$ . `gbayes1PowerNP` is a special case of `gbayesMixPowerNP` when the prior is a single Gaussian.

### Usage

```
gbayes(mean.prior, var.prior, m1, m2, stat, var.stat,
       n1, n2, cut.prior, cut.prob.prior=0.025)

## S3 method for class 'gbayes'
plot(x, xlim, ylim, name.stat='z', ...)

gbayes2(sd, prior, delta.w=0, alpha=0.05, upper=Inf, prior.aux)

gbayesMixPredNoData(mix=NA, d0=NA, v0=NA, d1=NA, v1=NA,
                    what=c('density', 'cdf'))

gbayesMixPost(x=NA, v=NA, mix=1, d0=NA, v0=NA, d1=NA, v1=NA,
              what=c('density', 'cdf', 'postmean'))

gbayesMixPowerNP(pcdf, delta, v, delta.w=0, mix, interval,
                 nsim=0, alpha=0.05)

gbayes1PowerNP(d0, v0, delta, v, delta.w=0, alpha=0.05)
```

### Arguments

<code>mean.prior</code>	mean of the prior distribution
<code>cut.prior, cut.prob.prior, var.prior</code>	variance of the prior. Use a large number such as 10000 to effectively use a flat (noninformative) prior. Sometimes it is useful to compute the variance so that the prior probability that <code>stat</code> is greater than some impressive value <code>u</code> is only <code>alpha</code> . The correct <code>var.prior</code> to use is then $((u - \text{mean.prior}) / \text{qnorm}(1 - \alpha))^2$ . You can specify <code>cut.prior=u</code> and <code>cut.prob.prior=alpha</code> (whose default is 0.025) in place of <code>var.prior</code> to have <code>gbayes</code> compute the prior variance in this manner.
<code>m1</code>	sample size in group 1
<code>m2</code>	sample size in group 2
<code>stat</code>	statistic comparing groups 1 and 2, e.g., log hazard ratio, difference in means, difference in angular transformations of proportions
<code>var.stat</code>	variance of <code>stat</code> , assumed to be known. <code>var.stat</code> should either be a constant (allowed if <code>n1</code> is not specified), or a function of two arguments which specify the sample sizes in groups 1 and 2. Calculations will be approximate when the variance is estimated from the data.
<code>x</code>	an object returned by <code>gbayes</code> or the value of the statistic which is an estimator of $\delta$ , the parameter of interest
<code>sd</code>	the standard deviation of the treatment effect

prior	a function of possibly a vector of unknown treatment effects, returning the prior density at those values
pcdf	a function computing the posterior CDF of the treatment effect <code>delta</code> , such as a function created by <code>gbayesMixPost</code> with <code>what="cdf"</code> .
delta	a true unknown single treatment effect to detect
v	the variance of the statistic <code>x</code> , e.g., $s^2 * (1/n1 + 1/n2)$ . Neither <code>x</code> nor <code>v</code> need to be defined to <code>gbayesMixPost</code> , as they can be defined at run time to the function created by <code>gbayesMixPost</code> .
n1	number of future observations in group 1, for obtaining a predictive distribution
n2	number of future observations in group 2
xlim	vector of 2 x-axis limits. Default is the mean of the posterior plus or minus 6 standard deviations of the posterior.
ylim	vector of 2 y-axis limits. Default is the range over combined prior and posterior densities.
name.stat	label for x-axis. Default is "z".
...	optional arguments passed to <code>labcurve</code> from <code>plot.gbayes</code>
delta.w	the minimum worthwhile treatment difference to detect. The default is zero for a plain uninteresting null hypothesis.
alpha	type I error, or more accurately one minus the confidence level for a two-sided confidence limit for the treatment effect
upper	upper limit of integration over the prior distribution multiplied by the normal likelihood for the treatment effect statistic. Default is infinity.
prior.aux	argument to pass to <code>prior</code> from <code>integrate</code> through <code>gbayes2</code> . Inside of <code>power</code> the argument must be named <code>prior.aux</code> if it exists. You can pass multiple parameters by passing <code>prior.aux</code> as a list and pulling off elements of the list inside <code>prior</code> . This setup was used because of difficulties in passing ... arguments through <code>integrate</code> for some situations.
mix	mixing probability or weight for the Gaussian prior having mean <code>d0</code> and variance <code>v0</code> . <code>mix</code> must be between 0 and 1, inclusive.
d0	mean of the first Gaussian distribution (only Gaussian for <code>gbayes1PowerNP</code> and is a required argument)
v0	variance of the first Gaussian (only Gaussian for <code>gbayes1PowerNP</code> and is a required argument)
d1	mean of the second Gaussian (if <code>mix &lt; 1</code> )
v1	variance of the second Gaussian (if <code>mix &lt; 1</code> ). Any of these last 5 arguments can be omitted to <code>gbayesMixPredNoData</code> as they can be provided at run time to the function created by <code>gbayesMixPredNoData</code> .
what	specifies whether the predictive density or the CDF is to be computed. Default is "density".
interval	a 2-vector containing the lower and upper limit for possible values of the test statistic <code>x</code> that would result in a left credible interval exceeding <code>delta.w</code> with probability $1-\alpha/2$

`nsim` defaults to zero, causing `gbayesMixPowerNP` to solve numerically for the critical value of  $x$ , then to compute the power accordingly. Specify a nonzero number such as 20000 for `nsim` to instead have the function estimate power by simulation. In this case 0.95 confidence limits on the estimated power are also computed. This approach is sometimes necessary if `uniroot` can't solve the equation for the critical value.

### Value

`gbayes` returns a list of class "gbayes" containing the following names elements: `mean.prior`, `var.prior`, `mean.post`, `var.post`, and if `n1` is specified, `mean.pred` and `var.pred`. Note that `mean.pred` is identical to `mean.post`. `gbayes2` returns a single number which is the probability of correctly rejecting the null hypothesis in favor of the new treatment. `gbayesMixPredNoData` returns a function that can be used to evaluate the predictive density or cumulative distribution. `gbayesMixPost` returns a function that can be used to evaluate the posterior density or cdf. `gbayesMixPowerNP` returns a vector containing two values if `nsim = 0`. The first value is the critical value for the test statistic that will make the left credible interval  $> \delta$ , and the second value is the power. If `nsim > 0`, it returns the power estimate and confidence limits for it if `nsim > 0`. The examples show how to use these functions.

### Author(s)

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University School of Medicine  
 <fh@fharrell.com>

### References

- Spiegelhalter DJ, Freedman LS, Parmar MKB (1994): Bayesian approaches to randomized trials. *JRSS A* 157:357–416. Results for `gbayes` are derived from Equations 1, 2, 3, and 6.
- Spiegelhalter DJ, Freedman LS (1986): A predictive approach to selecting the size of a clinical trial, based on subjective clinical opinion. *Stat in Med* 5:1–13.
- Joseph, Lawrence and Belisle, Patrick (1997): Bayesian sample size determination for normal means and differences between normal means. *The Statistician* 46:209–226.
- Grouin, JM, Coste M, Bunouf P, Lecoutre B (2007): Bayesian sample size determination in non-sequential clinical trials: Statistical aspects and some regulatory considerations. *Stat in Med* 26:4914–4924.

### See Also

[gbayesSeqSim](#)

### Examples

```
# Compare 2 proportions using the var stabilizing transformation
# arcsin(sqrt((x+3/8)/(n+3/4))) (Anscombe), which has variance
# 1/[4(n+.5)]
```



```

m1 <- 100;      m2 <- 150
deaths1 <- 10; deaths2 <- 30

f <- function(events,n) asin(sqrt((events+3/8)/(n+3/4)))
stat <- f(deaths1,m1) - f(deaths2,m2)
var.stat <- function(m1, m2) 1/4/(m1+.5) + 1/4/(m2+.5)
cat("Test statistic:",format(stat)," s.d.:",
    format(sqrt(var.stat(m1,m2))), "\n")
#Use unbiased prior with variance 1000 (almost flat)
b <- gbayes(0, 1000, m1, m2, stat, var.stat, 2*m1, 2*m2)
print(b)
plot(b)
#To get posterior Prob[parameter > w] use
# 1-pnorm(w, b$mean.post, sqrt(b$var.post))

#If g(effect, n1, n2) is the power function to
#detect an effect of 'effect' with samples size for groups 1 and 2
#of n1,n2, estimate the expected power by getting 1000 random
#draws from the posterior distribution, computing power for
#each value of the population effect, and averaging the 1000 powers
#This code assumes that g will accept vector-valued 'effect'
#For the 2-sample proportion problem just addressed, 'effect'
#could be taken approximately as the change in the arcsin of
#the square root of the probability of the event

g <- function(effect, n1, n2, alpha=.05) {
  sd <- sqrt(var.stat(n1,n2))
  z <- qnorm(1 - alpha/2)
  effect <- abs(effect)
  1 - pnorm(z - effect/sd) + pnorm(-z - effect/sd)
}

effects <- rnorm(1000, b$mean.post, sqrt(b$var.post))
powers <- g(effects, 500, 500)
hist(powers, nclass=35, xlab='Power')
describe(powers)

# gbayes2 examples
# First consider a study with a binary response where the
# sample size is n1=500 in the new treatment arm and n2=300
# in the control arm. The parameter of interest is the
# treated:control log odds ratio, which has variance
# 1/[n1 p1 (1-p1)] + 1/[n2 p2 (1-p2)]. This is not
# really constant so we average the variance over plausible
# values of the probabilities of response p1 and p2. We
# think that these are between .4 and .6 and we take a

```

```

# further short cut

v <- function(n1, n2, p1, p2) 1/(n1*p1*(1-p1)) + 1/(n2*p2*(1-p2))
n1 <- 500; n2 <- 300
ps <- seq(.4, .6, length=100)
vguess <- quantile(v(n1, n2, ps, ps), .75)
vguess
#           75%
# 0.02183459

# The minimally interesting treatment effect is an odds ratio
# of 1.1. The prior distribution on the log odds ratio is
# a 50:50 mixture of a vague Gaussian (mean 0, sd 100) and
# an informative prior from a previous study (mean 1, sd 1)

prior <- function(delta)
  0.5*dnorm(delta, 0, 100)+0.5*dnorm(delta, 1, 1)
deltas <- seq(-5, 5, length=150)
plot(deltas, prior(deltas), type='l')

# Now compute the power, averaged over this prior
gbayes2(sqrt(vguess), prior, log(1.1))
# [1] 0.6133338

# See how much power is lost by ignoring the previous
# study completely

gbayes2(sqrt(vguess), function(delta)dnorm(delta, 0, 100), log(1.1))
# [1] 0.4984588

# What happens to the power if we really don't believe the treatment
# is very effective? Let's use a prior distribution for the log
# odds ratio that is uniform between log(1.2) and log(1.3).
# Also check the power against a true null hypothesis

prior2 <- function(delta) dunif(delta, log(1.2), log(1.3))
gbayes2(sqrt(vguess), prior2, log(1.1))
# [1] 0.1385113

gbayes2(sqrt(vguess), prior2, 0)
# [1] 0.3264065

# Compare this with the power of a two-sample binomial test to

```

```

# detect an odds ratio of 1.25
bpower(.5, odds.ratio=1.25, n1=500, n2=300)
#   Power
# 0.3307486

# For the original prior, consider a new study with equal
# sample sizes n in the two arms. Solve for n to get a
# power of 0.9. For the variance of the log odds ratio
# assume a common p in the center of a range of suspected
# probabilities of response, 0.3. For this example we
# use a zero null value and the uniform prior above

v  <- function(n) 2/(n*.3*.7)
pow <- function(n) gbayes2(sqrt(v(n)), prior2)
uniroot(function(n) pow(n)-0.9, c(50,10000))$root
# [1] 2119.675
# Check this value
pow(2119.675)
# [1] 0.9

# Get the posterior density when there is a mixture of two priors,
# with mixing probability 0.5. The first prior is almost
# non-informative (normal with mean 0 and variance 10000) and the
# second has mean 2 and variance 0.3. The test statistic has a value
# of 3 with variance 0.4.
f <- gbayesMixPost(3, 4, mix=0.5, d0=0, v0=10000, d1=2, v1=0.3)

args(f)

# Plot this density
delta <- seq(-2, 6, length=150)
plot(delta, f(delta), type='l')

# Add to the plot the posterior density that used only
# the almost non-informative prior
lines(delta, f(delta, mix=1), lty=2)

# The same but for an observed statistic of zero
lines(delta, f(delta, mix=1, x=0), lty=3)

# Derive the CDF instead of the density
g <- gbayesMixPost(3, 4, mix=0.5, d0=0, v0=10000, d1=2, v1=0.3,
  what='cdf')
# Had mix=0 or 1, gbayes1PowerNP could have been used instead
# of gbayesMixPowerNP below

```

```

# Compute the power to detect an effect of delta=1 if the variance
# of the test statistic is 0.2
gbayesMixPowerNP(g, 1, 0.2, interval=c(-10,12))

# Do the same thing by simulation
gbayesMixPowerNP(g, 1, 0.2, interval=c(-10,12), nsim=20000)

# Compute by what factor the sample size needs to be larger
# (the variance needs to be smaller) so that the power is 0.9
ratios <- seq(1, 4, length=50)
pow <- single(50)
for(i in 1:50)
  pow[i] <- gbayesMixPowerNP(g, 1, 0.2/ratios[i], interval=c(-10,12))[2]

# Solve for ratio using reverse linear interpolation
approx(pow, ratios, xout=0.9)$y

# Check this by computing power
gbayesMixPowerNP(g, 1, 0.2/2.1, interval=c(-10,12))
# So the study will have to be 2.1 times as large as earlier thought

```

---

gbayesSeqSim

*gbayesSeqSim*


---

## Description

Simulate Bayesian Sequential Treatment Comparisons Using a Gaussian Model

## Usage

```
gbayesSeqSim(est, asserts)
```

## Arguments

<code>est</code>	data frame created by <code>estSeqSim()</code>
<code>asserts</code>	list of lists. The first element of each list is the user-specified name for each assertion/prior combination, e.g., "efficacy". The other elements are, in order, a character string equal to "<", ">", or "in", a parameter value cutoff (for "<" and ">") or a 2-vector specifying an interval for "in", and either a prior distribution mean and standard deviation named <code>mu</code> and <code>sigma</code> respectively, or a parameter value (" <code>cutprior</code> ") and tail area " <code>tailprob</code> ". If the latter is used, <code>mu</code> is assumed to be zero and <code>sigma</code> is solved for such that $P(\text{parameter} > \text{'cutprior'}) = P(\text{parameter} < - \text{'cutprior'}) = \text{tailprob}$ .

**Details**

Simulate a sequential trial under a Gaussian model for parameter estimates, and Gaussian priors using simulated estimates and variances returned by `estSeqSim`. For each row of the data frame `est` and for each prior/assertion combination, computes the posterior probability of the assertion.

**Value**

a data frame with number of rows equal to that of `est` with a number of new columns equal to the number of assertions added. The new columns are named `p1`, `p2`, `p3`, ... (posterior probabilities), `mean1`, `mean2`, ... (posterior means), and `sd1`, `sd2`, ... (posterior standard deviations). The returned data frame also has an attribute `asserts` added which is the original `asserts` augmented with any derived `mu` and `sigma` and converted to a data frame, and another attribute `alabels` which is a named vector used to map `p1`, `p2`, ... to the user-provided labels in `asserts`.

**Author(s)**

Frank Harrell

**See Also**

`gbayes()`, `estSeqSim()`, `simMarkovOrd()`, `estSeqMarkovOrd()`

**Examples**

```
## Not run:
# Simulate Bayesian operating characteristics for an unadjusted
# proportional odds comparison (Wilcoxon test)
# For 100 simulations, 5 looks, 2 true parameter values, and
# 2 assertion/prior combinations, compute the posterior probability
# Use a low-level logistic regression call to speed up simulations
# Use data.table to compute various summary measures
# Total simulation time: 2s
lfit <- function(x, y) {
  f <- rms::lrn.fit(x, y)
  k <- length(coef(f))
  c(coef(f)[k], vcov(f)[k, k])
}
gdat <- function(beta, n1, n2) {
  # Cell probabilities for a 7-category ordinal outcome for the control group
  p <- c(2, 1, 2, 7, 8, 38, 42) / 100

  # Compute cell probabilities for the treated group
  p2 <- pomodm(p=p, odds.ratio=exp(beta))
  y1 <- sample(1 : 7, n1, p, replace=TRUE)
  y2 <- sample(1 : 7, n2, p2, replace=TRUE)
  list(y1=y1, y2=y2)
}

# Assertion 1: log(OR) < 0 under prior with prior mean 0.1 and sigma 1 on log OR scale
# Assertion 2: OR between 0.9 and 1/0.9 with prior mean 0 and sigma computed so that
# P(OR > 2) = 0.05
```

```

asserts <- list(list('Efficacy', '<', 0, mu=0.1, sigma=1),
               list('Similarity', 'in', log(c(0.9, 1/0.9)),
                   cutprior=log(2), tailprob=0.05))

set.seed(1)
est <- estSeqSim(c(0, log(0.7)), looks=c(50, 75, 95, 100, 200),
               gendat=gdat,
               fitter=lfit, nsim=100)
z <- gbayesSeqSim(est, asserts)
head(z)
attr(z, 'asserts')

# Compute the proportion of simulations that hit targets (different target posterior
# probabilities for efficacy vs. similarity)

# For the efficacy assessment compute the first look at which the target
# was hit (set to infinity if never hit)
require(data.table)
z <- data.table(z)
u <- z[, .(first=min(p1 > 0.95)), by=.(parameter, sim)]
# Compute the proportion of simulations that ever hit the target and
# that hit it by the 100th subject
u[, .(ever=mean(first < Inf)), by=.(parameter)]
u[, .(by75=mean(first <= 100)), by=.(parameter)]

## End(Not run)

```

---

getabd

*getabd*


---

## Description

Data from The Analysis of Biological Data by Shitlock and Schluter

## Usage

```
getabd(name = "", lowernames = FALSE, allow = "_")
```

## Arguments

name	name of dataset to fetch. Omit to get a data table listing all available datasets.
lowernames	set to TRUE to change variable names to lower case
allow	set to NULL to convert underscores in variable names to periods

## Details

Fetches csv files for exercises in the book

**Value**

data frame with attributes label and url

**Author(s)**

Frank Harrell

---

getHdata	<i>Download and Install Datasets for <b>Hmisc</b>, <b>rms</b>, and Statistical Modeling</i>
----------	---

---

**Description**

This function downloads and makes ready to use datasets from the main web site for the **Hmisc** and **rms** libraries. For **R**, the datasets were stored in compressed [save](#) format and getHdata makes them available by running [load](#) after download. For S-Plus, the datasets were stored in data.dump format and are made available by running data.restore after import. The dataset is run through the [cleanup.import](#) function. Calling getHdata with no file argument provides a character vector of names of available datasets that are currently on the web site. For **R**, **R**'s default browser can optionally be launched to view html files that were already prepared using the **Hmisc** command `html(contents())` or to view '.txt' or '.html' data description files when available.

If `options(localHfiles=TRUE)` the scripts are read from local directory `~/web/data/repo` instead of from the web server.

**Usage**

```
getHdata(file, what = c("data", "contents", "description", "all"),
         where="https://hbiostat.org/data/repo")
```

**Arguments**

file	an unquoted name of a dataset on the web site, e.g. 'prostate'. Omit file to obtain a list of available datasets.
what	specify <code>what="contents"</code> to browse the contents (metadata) for the dataset rather than fetching the data themselves. Specify <code>what="description"</code> to browse a data description file if available. Specify <code>what="all"</code> to retrieve the data and see the metadata and description.
where	URL containing the data and metadata files

**Value**

`getHdata()` without a file argument returns a character vector of dataset base names. When a dataset is downloaded, the data frame is placed in search position one and is not returned as value of `getHdata`.

**Author(s)**

Frank Harrell

**See Also**[download.file](#), [cleanup.import](#), [data.restore](#), [load](#)**Examples**

```
## Not run:
getHdata()           # download list of available datasets
getHdata(prostate)  # downloads, load( ) or data.restore( )
                    # runs cleanup.import for S-Plus 6
getHdata(valung, "contents") # open browser (options(browser="whatever"))
                    # after downloading valung.html
                    # (result of html(contents()))
getHdata(support, "all") # download and open one browser window
datadensity(support)
attach(support)      # make individual variables available
getHdata(plasma, "all") # download and open two browser windows
                    # (description file is available for plasma)

## End(Not run)
```

getRs

*Interact with github rscripts Project***Description**

The github rscripts project at <https://github.com/harrelfe/rscripts> contains R scripts that are primarily analysis templates for teaching with RStudio. This function allows the user to print an organized list of available scripts, to download a script and `source()` it into the current session (the default), to download a script and load it into an RStudio script editor window, to list scripts whose major category contains a given string (ignoring case), or to list all major and minor categories. If `options(localHfiles=TRUE)` the scripts are read from local directory `~/R/rscripts` instead of from github.

**Usage**

```
getRs(file=NULL, guser='harrelfe', grepo='rscripts', gdir='raw/master',
      dir=NULL, browse=c('local', 'browser'), cats=FALSE,
      put=c('source', 'rstudio'))
```

**Arguments**

`file` a character string containing a script file name. Omit `file` to obtain a list of available scripts with major and minor categories.

`guser` GitHub user name, default is 'harrelfe'



grepo	Github repository name, default is 'rscripts'
gdir	Github directory under which to find retrievable files
dir	directory under grepo in which to find files
browse	When showing the rscripts contents directory, the default is to list in tabular form in the console. Specify browse='browser' to open the online contents in a web browser.
cats	Leave at the default (FALSE) to list whole contents or download a script. Specify cats=TRUE to list major and minor categories available. Specify a character string to list all scripts whose major category contains the string (ignoring case).
put	Leave at the default ('source') to source() the file. This is useful when the file just defines a function you want to use in the session. Use load put='rstudio' to load the file into the RStudio script editor window using the rstudioapi navigateToFile function. If RStudio is not running, file.edit() is used instead.

**Value**

a data frame or list, depending on arguments

**Author(s)**

Frank Harrell and Cole Beck

**See Also**

[download.file](#)

**Examples**

```
## Not run:
getRs()           # list available scripts
scripts <- getRs() # likewise, but store in an object that can easily
                  # be viewed on demand in RStudio
getRs('introda.r') # download introda.r and put in script editor
getRs(cats=TRUE)   # list available major and minor categories
categories <- getRs(cats=TRUE)
# likewise but store results in a list for later viewing
getRs(cats='reg') # list all scripts in a major category containing 'reg'
getRs('importREDCap.r') # source() to define a function
# source() a new version of the Hmisc package's cut2 function:
getRs('cut2.s', grepo='Hmisc', dir='R')

## End(Not run)
```

---

`getZip`*Open a Zip File From a URL*

---

**Description**

Allows downloading and reading of a zip file containing one file

**Usage**

```
getZip(url, password=NULL)
```

**Arguments**

<code>url</code>	either a path to a local file or a valid URL.
<code>password</code>	required to decode password-protected zip files

**Details**

Allows downloading and reading of zip file containing one file. The file may be password protected. If a password is needed then one will be requested unless given.

Note: to make password-protected zip file z.zip, do `zip -e z myfile`

**Value**

Returns a file O/I pipe.

**Author(s)**

Frank E. Harrell

**See Also**

[pipe](#)

**Examples**

```
## Not run:  
read.csv(getZip('http://test.com/z.zip'))  
  
## End(Not run)
```

**Description**

Uses `ggplot2` to plot a scatterplot or dot-like chart for the case where there is a very large number of overlapping values. This works for continuous and categorical `x` and `y`. For continuous variables it serves the same purpose as hexagonal binning. Counts for overlapping points are grouped into quantile groups and level of transparency and rainbow colors are used to provide count information.

Instead, you can specify `stick=TRUE` not use color but to encode cell frequencies with the height of a black line `y`-centered at the middle of the bins. Relative frequencies are not transformed, and the maximum cell frequency is shown in a caption. Every point with at least a frequency of one is depicted with a full-height light gray vertical line, scaled to the above overall maximum frequency. In this way to relative frequency is to proportion of these light gray lines that are black, and one can see points whose frequencies are too low to see the black lines.

The result can also be passed to `ggplotly`. Actual cell frequencies are added to the hover text in that case using the `label ggplot2` aesthetic.

**Usage**

```
ggfreqScatter(x, y, by=NULL, bins=50, g=10, cuts=NULL,
              xtrans = function(x) x,
              ytrans = function(y) y,
              xbreaks = pretty(x, 10),
              ybreaks = pretty(y, 10),
              xminor = NULL, yminor = NULL,
              xlab = as.character(substitute(x)),
              ylab = as.character(substitute(y)),
              fcolors = viridis::viridis(10), nsize=FALSE,
              stick=FALSE, html=FALSE, prfreq=FALSE, ...)
```

**Arguments**

<code>x</code>	x-variable
<code>y</code>	y-variable
<code>by</code>	an optional vector used to make separate plots for each distinct value using <code>facet_wrap()</code>
<code>bins</code>	for continuous <code>x</code> or <code>y</code> is the number of bins to create by rounding. Ignored for categorical variables. If a 2-vector, the first element corresponds to <code>x</code> and the second to <code>y</code> .
<code>g</code>	number of quantile groups to make for frequency counts. Use <code>g=0</code> to use frequencies continuously for color coding. This is recommended only when using <code>plotly</code> .
<code>cuts</code>	instead of using <code>g</code> , specify <code>cuts</code> to provide the vector of cuts for categorizing frequencies for assignment to colors

xtrans, ytrans	functions specifying transformations to be made before binning and plotting
xbreaks, ybreaks	vectors of values to label on axis, on original scale
xminor, yminor	values at which to put minor tick marks, on original scale
xlab, ylab	axis labels. If not specified and variable has a label, that label will be used.
fcolors	colors argument to pass to <code>scale_color_gradientn</code> to color code frequencies. Use <code>fcolors=gray.colors(10, 0.75, 0)</code> to show gray scale, for example. Another good choice is <code>fcolors=hcl.colors(10, 'Blue-Red')</code> .
nsize	set to TRUE to not vary color or transparency but instead to size the symbols in relation to the number of points. Best with both x and y are discrete. <code>ggplot2</code> size is taken as the fourth root of the frequency. If there are 15 or unique frequencies all the unique frequencies are used, otherwise g quantile groups of frequencies are used.
stick	set to TRUE to not use colors but instead use varying-height black vertical lines to depict cell frequencies.
html	set to TRUE to use html in axis labels instead of plotmath
prfreq	set to TRUE to print the frequency distributions of the binned coordinate frequencies
...	arguments to pass to <code>geom_point</code> such as shape and size

**Value**

a `ggplot` object

**Author(s)**

Frank Harrell

**See Also**

[cut2](#)

**Examples**

```
require(ggplot2)
set.seed(1)
x <- rnorm(1000)
y <- rnorm(1000)
count <- sample(1:100, 1000, TRUE)
x <- rep(x, count)
y <- rep(y, count)
# color=alpha=NULL below makes loess smooth over all points
g <- ggfreqScatter(x, y) + # might add g=0 if using plotly
  geom_smooth(aes(color=NULL, alpha=NULL), se=FALSE) +
  ggtitle("Using Deciles of Frequency Counts, 2500 Bins")
g
# plotly::ggplotly(g, tooltip='label') # use plotly, hover text = freq. only
# Plotly makes it somewhat interactive, with hover text tooltips
```

```

# Instead use varying-height sticks to depict frequencies
ggfreqScatter(x, y, stick=TRUE) +
  labs(subtitle='Relative height of black lines to gray lines
is proportional to cell frequency.
Note that points with even tiny frequency are visible
(gray line with no visible black line).')

# Try with x categorical
x1 <- sample(c('cat', 'dog', 'giraffe'), length(x), TRUE)
ggfreqScatter(x1, y)

# Try with y categorical
y1 <- sample(LETTERS[1:10], length(x), TRUE)
ggfreqScatter(x, y1)

# Both categorical, larger point symbols, box instead of circle
ggfreqScatter(x1, y1, shape=15, size=7)
# Vary box size instead
ggfreqScatter(x1, y1, nsize=TRUE, shape=15)

```

---

ggplotlyr

*ggplotlyr*


---

## Description

Render plotly Graphic from a ggplot2 Object

## Usage

```
ggplotlyr(ggobject, tooltip = "label", remove = "txt: ", ...)
```

## Arguments

ggobject	an object produced by ggplot
tooltip	attribute specified to ggplot to hold hover text
remove	extraneous text to remove from hover text. Default is set to assume tooltip='label' and assumed the user specified aes(..., label=txt). If you instead specified aes(..., label=myvar) use remove='myvar: '.
...	other arguments passed to ggplotly

## Details

Uses `plotly::ggplotly()` to render a plotly graphic with a specified tooltip attribute, removing extraneous text that ggplotly puts in hover text when tooltip='label'

**Value**

a plotly object

**Author(s)**

Frank Harrell

---

GiniMd

*Gini's Mean Difference*

---

**Description**

GiniMD computes Gini's mean difference on a numeric vector. This index is defined as the mean absolute difference between any two distinct elements of a vector. For a Bernoulli (binary) variable with proportion of ones equal to  $p$  and sample size  $n$ , Gini's mean difference is  $2\frac{n}{n-1}p(1-p)$ . For a trinomial variable (e.g., predicted values for a 3-level categorical predictor using two dummy variables) having (predicted) values  $A, B, C$  with corresponding proportions  $a, b, c$ , Gini's mean difference is  $2\frac{n}{n-1}[ab|A-B| + ac|A-C| + bc|B-C|]$

**Usage**

```
GiniMd(x, na.rm=FALSE)
```

**Arguments**

<code>x</code>	a numeric vector (for GiniMd)
<code>na.rm</code>	set to TRUE if you suspect there may be NAs in <code>x</code> ; these will then be removed. Otherwise an error will result.

**Value**

a scalar numeric

**Author(s)**

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>

**References**

David HA (1968): Gini's mean difference rediscovered. *Biometrika* 55:573–575.

**Examples**

```

set.seed(1)
x <- rnorm(40)
# Test GiniMd against a brute-force solution
gmd <- function(x) {
  n <- length(x)
  sum(outer(x, x, function(a, b) abs(a - b))) / n / (n - 1)
}
GiniMd(x)
gmd(x)

z <- c(rep(0,17), rep(1,6))
n <- length(z)
GiniMd(z)
2*mean(z)*(1-mean(z))*n/(n-1)

a <- 12; b <- 13; c <- 7; n <- a + b + c
A <- -.123; B <- -.707; C <- 0.523
xx <- c(rep(A, a), rep(B, b), rep(C, c))
GiniMd(xx)
2*(a*b*abs(A-B) + a*c*abs(A-C) + b*c*abs(B-C))/n/(n-1)

```

hashCheck

*hashCheck***Description**

Check for Changes in List of Objects

**Usage**

```
hashCheck(..., file, .print. = TRUE, .names. = NULL)
```

**Arguments**

...	a list of objects including data frames, vectors, functions, and all other types of R objects that represent dependencies of a certain calculation
file	name of file in which results are stored
.print.	set to FALSE to suppress printing information messages about what has changed
.names.	vector of names of original arguments if not calling hashCheck directly

**Details**

Given an RDS file name and a list of objects, does the following:

- makes a vector of hashes, one for each object. Function objects are run through deparse so that the environment of the function will not be considered.

- see if the file exists; if not, return a list with result=NULL, hash = new vector of hashes, changed='All'
- if the file exists, read the file and its hash attribute as prevhash
- if prevhash is not identical to hash: if .print.=TRUE (default), print to console a summary of what's changed return a list with result=NULL, hash = new hash vector, changed
- if prevhash = hash, return a list with result=file object, hash=new hash, changed=""

Set options(debughash=TRUE) to trace results in /tmp/debughash.txt

### Value

a list with elements result (the computations), hash (the new hash), and changed which details what changed to make computations need to be run

### Author(s)

Frank Harrell

---

hdquantile

*Harrell-Davis Distribution-Free Quantile Estimator*

---

### Description

Computes the Harrell-Davis (1982) quantile estimator and jackknife standard errors of quantiles. The quantile estimator is a weighted linear combination of order statistics in which the order statistics used in traditional nonparametric quantile estimators are given the greatest weight. In small samples the H-D estimator is more efficient than traditional ones, and the two methods are asymptotically equivalent. The H-D estimator is the limit of a bootstrap average as the number of bootstrap resamples becomes infinitely large.

### Usage

```
hdquantile(x, probs = seq(0, 1, 0.25),
           se = FALSE, na.rm = FALSE, names = TRUE, weights=FALSE)
```

### Arguments

x	a numeric vector
probs	vector of quantiles to compute
se	set to TRUE to also compute standard errors
na.rm	set to TRUE to remove NAs from x before computing quantiles
names	set to FALSE to prevent names attributions from being added to quantiles and standard errors
weights	set to TRUE to return a "weights" attribution with the matrix of weights used in the H-D estimator corresponding to order statistics, with columns corresponding to quantiles.



**Details**

A Fortran routine is used to compute the jackknife leave-out-one quantile estimates. Standard errors are not computed for quantiles 0 or 1 (NAs are returned).

**Value**

A vector of quantiles. If `se=TRUE` this vector will have an attribute `se` added to it, containing the standard errors. If `weights=TRUE`, also has a "weights" attribute which is a matrix.

**Author(s)**

Frank Harrell

**References**

Harrell FE, Davis CE (1982): A new distribution-free quantile estimator. *Biometrika* 69:635-640.

Hutson AD, Ernst MD (2000): The exact bootstrap mean and variance of an L-estimator. *J Roy Statist Soc B* 62:89-94.

**See Also**

[quantile](#)

**Examples**

```
set.seed(1)
x <- runif(100)
hdquantile(x, (1:3)/4, se=TRUE)

## Not run:
# Compare jackknife standard errors with those from the bootstrap
library(boot)
boot(x, function(x,i) hdquantile(x[i], probs=(1:3)/4), R=400)

## End(Not run)
```

**Description**

Moving and hiding table of contents for Rmd HTML documents

**Usage**

```

hidingTOC(
  buttonLabel = "Contents",
  levels = 3,
  tocSide = c("right", "left"),
  buttonSide = c("right", "left"),
  posCollapse = c("margin", "top", "bottom"),
  hidden = FALSE
)

```

**Arguments**

buttonLabel	the text on the button that hides and unhides the table of contents. Defaults to Contents.
levels	the max depth of the table of contents that it is desired to have control over the display of. (defaults to 3)
tocSide	which side of the page should the table of contents be placed on. Can be either 'right' or 'left'. Defaults to 'right'
buttonSide	which side of the page should the button that hides the TOC be placed on. Can be either 'right' or 'left'. Defaults to 'right'
posCollapse	if 'margin' then display the depth select buttons vertically along the side of the page choosen by buttonSide. If 'top' then display the depth select buttons horizontally under the button that hides the TOC. Defaults to 'margin'. 'bottom' is currently unimplemented.
hidden	Logical should the table of contents be hidden at page load Defaults to FALSE

**Details**

hidingTOC creates a table of contents in a Rmd document that can be hidden at the press of a button. It also generate buttons that allow the hiding or unhiding of the diffrent level depths of the table of contents.

**Value**

a HTML formatted text string to be inserted into an markdown document

**Author(s)**

Thomas Dupont

**Examples**

```

## Not run:
hidingTOC()

## End(Not run)

```

---

hist.data.frame      *Histograms for Variables in a Data Frame*

---

### Description

This functions tries to compute the maximum number of histograms that will fit on one page, then it draws a matrix of histograms. If there are more qualifying variables than will fit on a page, the function waits for a mouse click before drawing the next page.

### Usage

```
## S3 method for class 'data.frame'  
hist(x, n.unique = 3, nclass = "compute",  
      na.big = FALSE, rugs = FALSE, freq=TRUE, mtitl = FALSE, ...)
```

### Arguments

x	a data frame
n.unique	minimum number of unique values a variable must have before a histogram is drawn
nclass	number of bins. Default is $\max(2, \text{trunc}(\min(n/10, 25 * \log(n, 10))/2))$ , where n is the number of non-missing values for a variable.
na.big	set to TRUE to draw the number of missing values on the top of the histogram in addition to in a subtitle. In the subtitle, n is the number of non-missing values and m is the number of missing values
rugs	set to TRUE to add rug plots at the top of each histogram
freq	see <a href="#">hist</a> . Default is to show frequencies.
mtitl	set to a character string to set aside extra outside top margin and to use the string for an overall title
...	arguments passed to <a href="#">scat1d</a>

### Value

the number of pages drawn

### Author(s)

Frank E Harrell Jr

### See Also

[hist](#), [scat1d](#)

**Examples**

```
d <- data.frame(a=runif(200), b=rnorm(200),
               w=factor(sample(c('green','red','blue'), 200, TRUE)))
hist.data.frame(d) # in R, just say hist(d)
```

---

 histbackback

*Back to Back Histograms*


---

**Description**

Takes two vectors or a list with x and y components, and produces back to back histograms of the two datasets.

**Usage**

```
histbackback(x, y, brks=NULL, xlab=NULL, axes=TRUE, probability=FALSE,
            xlim=NULL, ylab='', ...)
```

**Arguments**

x, y	either two vectors or a list given as x with two components. If the components have names, they will be used to label the axis (modification FEH).
brks	vector of the desired breakpoints for the histograms.
xlab	a vector of two character strings naming the two datasets.
axes	logical flag stating whether or not to label the axes.
probability	logical flag: if TRUE, then the x-axis corresponds to the units for a density. If FALSE, then the units are counts.
xlim	x-axis limits. First value must be negative, as the left histogram is placed at negative x-values. Second value must be positive, for the right histogram. To make the limits symmetric, use e.g. ylim=c(-20,20).
ylab	label for y-axis. Default is no label.
...	additional graphics parameters may be given.

**Value**

a list is returned invisibly with the following components:

left	the counts for the dataset plotted on the left.
right	the counts for the dataset plotted on the right.
breaks	the breakpoints used.

**Side Effects**

a plot is produced on the current graphics device.

**Author(s)**

Pat Burns  
 Salomon Smith Barney  
 London  
 <pburns@dorado.sbi.com>

**See Also**

[hist](#), [histogram](#)

**Examples**

```
options(digits=3)
set.seed(1)
histbackback(rnorm(20), rnorm(30))

fool <- list(x=rnorm(40), y=rnorm(40))
histbackback(fool)
age <- rnorm(1000,50,10)
sex <- sample(c('female','male'),1000,TRUE)
histbackback(split(age, sex))
agef <- age[sex=='female'];agem <- age[sex=='male']
histbackback(list(Female=agef,Male=agem), probability=TRUE, xlim=c(-.06,.06))
```

---

 histboxp

---

*Use plotly to Draw Stratified Spike Histogram and Box Plot Statistics*


---

**Description**

Uses plotly to draw horizontal spike histograms stratified by group, plus the mean (solid dot) and vertical bars for these quantiles: 0.05 (red, short), 0.25 (blue, medium), 0.50 (black, long), 0.75 (blue, medium), and 0.95 (red, short). The robust dispersion measure Gini's mean difference and the SD may optionally be added. These are shown as horizontal lines starting at the minimum value of  $x$  having a length equal to the mean difference or SD. Even when Gini's and SD are computed, they are not drawn unless the user clicks on their legend entry.

Spike histograms have the advantage of effectively showing the raw data for both small and huge datasets, and unlike box plots allow multi-modality to be easily seen.

histboxpM plots multiple histograms stacked vertically, for variables in a data frame having a common group variable (if any) and combined using plotly::subplot.

dhistboxp is like histboxp but no plotly graphics are actually drawn. Instead, a data frame suitable for use with plotlyM is returned. For dhistboxp an additional level of stratification strata is implemented. group causes a different result here to produce back-to-back histograms (in the case of two groups) for each level of strata.

**Usage**

```

histboxp(p = plotly::plot_ly(height=height), x, group = NULL,
         xlab=NULL, gmd=TRUE, sd=FALSE, bins = 100, wmax=190, mult=7,
         connect=TRUE, showlegend=TRUE)

dhistboxp(x, group = NULL, strata=NULL, xlab=NULL,
          gmd=FALSE, sd=FALSE, bins = 100, nmin=5, ff1=1, ff2=1)

histboxpM(p=plotly::plot_ly(height=height, width=width), x, group=NULL,
          gmd=TRUE, sd=FALSE, width=NULL, nrows=NULL, ncols=NULL, ...)

```

**Arguments**

<code>p</code>	plotly graphics object if already begun
<code>x</code>	a numeric vector, or for <code>histboxpM</code> a numeric vector or a data frame of numeric vectors, hopefully with <code>label</code> and <code>units</code> attributes
<code>group</code>	a discrete grouping variable. If omitted, defaults to a vector of ones
<code>strata</code>	a discrete numeric stratification variable. Values are also used to space out different spike histograms. Defaults to a vector of ones.
<code>xlab</code>	x-axis label, defaults to labelled version include units of measurement if any
<code>gmd</code>	set to <code>FALSE</code> to not compute Gini's mean difference
<code>sd</code>	set to <code>TRUE</code> to compute the SD
<code>width</code>	width in pixels
<code>nrows</code>	number of rows for layout of multiple plots
<code>ncols</code>	number of columns for layout of multiple plots. At most one of <code>nrows</code> , <code>ncols</code> should be specified.
<code>bins</code>	number of equal-width bins to use for spike histogram. If the number of distinct values of <code>x</code> is less than <code>bins</code> , the actual values of <code>x</code> are used.
<code>nmin</code>	minimum number of non-missing observations for a group-stratum combination before the spike histogram and quantiles are drawn
<code>ff1, ff2</code>	fudge factors for position and bar length for spike histograms
<code>wmax, mult</code>	tweaks for margin to allocate
<code>connect</code>	set to <code>FALSE</code> to suppress lines connecting quantiles
<code>showlegend</code>	used if producing multiple plots to be combined with <code>subplot</code> ; set to <code>FALSE</code> for all but one plot
<code>...</code>	other arguments for <code>histboxpM</code> that are passed to <code>histboxp</code>

**Value**

a plotly object. For `dhistboxp` a data frame as expected by `plotlyM`

**Author(s)**

Frank Harrell

**See Also**

[histSpike](#), [plot.describe](#), [scat1d](#)

**Examples**

```
## Not run:
dist <- c(rep(1, 500), rep(2, 250), rep(3, 600))
Distribution <- factor(dist, 1 : 3, c('Unimodal', 'Bimodal', 'Trimodal'))
x <- c(rnorm(500, 6, 1),
      rnorm(200, 3, .7), rnorm(50, 7, .4),
      rnorm(200, 2, .7), rnorm(300, 5.5, .4), rnorm(100, 8, .4))
histboxp(x=x, group=Distribution, sd=TRUE)
X <- data.frame(x, x2=runif(length(x)))
histboxpM(x=X, group=Distribution, ncols=2) # separate plots

## End(Not run)
```

---

hlab

*hlab*

---

**Description**

Easy Extraction of Labels/Units Expressions for Plotting

**Usage**

```
hlab(x, name = NULL, html = FALSE, plotmath = TRUE)
```

**Arguments**

x	a single variable name, unquoted
name	a single character string providing an alternate way to name x that is useful when hlab is called from another function such as hlabs
html	set to TRUE to return HTML strings instead of plotmath expressions
plotmath	set to FALSE to use plain text instead of plotmath

**Details**

Given a single unquoted variable, first looks to see if a non-NULL LabelsUnits object exists (produced by `extractlabs()`). When LabelsUnits does not exist or is NULL, looks up the attributes in the current dataset, which defaults to `d` or may be specified by options(`current_ds='name of the data frame/table'`). Finally the existence of a variable of the given name in the global environment is checked. When a variable is not found in any of these three sources or has a blank label and units, an `expression()` with the variable name alone is returned. If `html=TRUE`, HTML strings are constructed instead, suitable for `plotly` graphics.

The result is useful for `xlab` and `ylab` in base plotting functions or in `ggplot2`, along with being useful for `labs` in `ggplot2`. See example.

**Value**

an expression created by `labelPlotmath` with `plotmath=TRUE`

**Author(s)**

Frank Harrell

**See Also**

[label\(\)](#), [units\(\)](#), [contents\(\)](#), [hlabs\(\)](#), [extractlabs\(\)](#), [plotmath](#)

**Examples**

```
d <- data.frame(x=1:10, y=(1:10)/10)
d <- upData(d, labels=c(x='X', y='Y'), units=c(x='mmHg'), print=FALSE)
hlab(x)
hlab(x, html=TRUE)
hlab(z)
require(ggplot2)
ggplot(d, aes(x, y)) + geom_point() + labs(x=hlab(x), y=hlab(y))
# Can use xlab(hlab(x)) + ylab(hlab(y)) also
# Store names, labels, units for all variables in d in object
LabelsUnits <- extractlabs(d)
# Remove d; labels/units still found
rm(d)
hlab(x)
# Remove LabelsUnits and use a current dataset named
# d2 instead of the default d
rm(LabelsUnits)
options(current_ds='d2')
```

---

hlabs

*hlabs*

---

**Description**

Front-end to `ggplot2` `labs` Function

**Usage**

```
hlabs(x, y, html = FALSE)
```

**Arguments**

<code>x</code>	a single variable name, unquoted
<code>y</code>	a single variable name, unquoted
<code>html</code>	set to <code>TRUE</code> to render in html (for <code>plotly</code> ), otherwise the result is <code>plotmath</code> expressions



**Details**

Runs `x`, `y`, or both through `hlab()` and passes the constructed labels to the `ggplot2::labs` function to specify `x`- and `y`-axis labels specially formatted for units of measurement

**Value**

result of `ggplot2::labs()`

**Author(s)**

Frank Harrell

**Examples**

```
# Name the current dataset d, or specify a name with
# options(curr_ds='...') or run `extractlabs`, then
# ggplot(d, aes(x,y)) + geom_point() + hlabs(x,y)
# to specify only the x-axis label use hlabs(x), or to
# specify only the y-axis label use hlabs(y=...)
```

**Description**

The Hmisc library contains many functions useful for data analysis, high-level graphics, utility operations, functions for computing sample size and power, translating SAS datasets into R, imputing missing values, advanced table making, variable clustering, character string manipulation, conversion of R objects to LaTeX code, recoding variables, and bootstrap repeated measures analysis. Most of these functions were written by F Harrell, but a few were collected from statlib and from s-news; other authors are indicated below. This collection of functions includes all of Harrell's submissions to statlib other than the functions in the `rms` and `display` libraries. A few of the functions do not have "Help" documentation.

To make **Hmisc** load silently, issue `options(Hverbose=FALSE)` before `library(Hmisc)`.

**Functions**

<b>Function Name</b>	<b>Purpose</b>
<code>abs.error.pred</code>	Computes various indexes of predictive accuracy based on absolute errors, for linear models
<code>addMarginal</code>	Add marginal observations over selected variables
<code>all.is.numeric</code>	Check if character strings are legal numerics
<code>approxExtrap</code>	Linear extrapolation
<code>aregImpute</code>	Multiple imputation based on additive regression, bootstrapping, and predictive mean matching
<code>areg.boot</code>	Nonparametrically estimate transformations for both

	sides of a multiple additive regression, and bootstrap these estimates and $R^2$
ballocation	Optimum sample allocations in 2-sample proportion test
binconf	Exact confidence limits for a proportion and more accurate (narrower!) score stat.-based Wilson interval (Rollin Brant, mod. FEH)
bootkm	Bootstrap Kaplan-Meier survival or quantile estimates
bpower	Approximate power of 2-sided test for 2 proportions Includes bpower.sim for exact power by simulation
bpplot	Box-Percentile plot (Jeffrey Banfield, <umsfjban@bill.oscs.montana.edu>)
bpplotM	Chart extended box plots for multiple variables
bsamsize	Sample size requirements for test of 2 proportions
bystats	Statistics on a single variable by levels of $\geq 1$ factors
bystats2	2-way statistics
character.table	Shows numeric equivalents of all latin characters Useful for putting many special chars. in graph titles (Pierre Joyet, <pierre.joyet@bluewin.ch>)
ciapower	Power of Cox interaction test
cleanup.import	More compactly store variables in a data frame, and clean up problem data when e.g. Excel spreadsheet had a non-numeric value in a numeric column
combine.levels	Combine infrequent levels of a categorical variable
confbar	Draws confidence bars on an existing plot using multiple confidence levels distinguished using color or gray scale
contents	Print the contents (variables, labels, etc.) of a data frame
cpower	Power of Cox 2-sample test allowing for noncompliance
Cs	Vector of character strings from list of unquoted names
csv.get	Enhanced importing of comma separated files labels
cut2	Like cut with better endpoint label construction and allows construction of quantile groups or groups with given n
datadensity	Snapshot graph of distributions of all variables in a data frame. For continuous variables uses scat1d.
dataRep	Quantify representation of new observations in a database
ddmmyy	SAS "date7" output format for a chron object
deff	Kish design effect and intra-cluster correlation
describe	Function to describe different classes of objects. Invoke by saying describe(object). It calls one of the following:
describe.data.frame	Describe all variables in a data frame (generalization of SAS UNIVARIATE)
describe.default	Describe a variable (generalization of SAS UNIVARIATE)
dotplot3	A more flexible version of dotplot
Dotplot	Enhancement of Trellis dotplot allowing for matrix x-var., auto generation of Key function, superposition
drawPlot	Simple mouse-driven drawing program, including a function for fitting Bezier curves
Ecdf	Empirical cumulative distribution function plot

errbar	Plot with error bars (Charles Geyer, U. Chi., mod FEH)
event.chart	Plot general event charts (Jack Lee, <jjlee@mdanderson.org>, Ken Hess, Joel Dubin; Am Statistician 54:63-70,2000)
event.history	Event history chart with time-dependent cov. status (Joel Dubin, <jdubin@uwaterloo.ca>)
find.matches	Find matches (with tolerances) between columns of 2 matrices
first.word	Find the first word in an R expression (R Heiberger)
fit.mult.impute	Fit most regression models over multiple transcan imputations, compute imputation-adjusted variances and avg. betas
format.df	Format a matrix or data frame with much user control (R Heiberger and FE Harrell)
ftupwr	Power of 2-sample binomial test using Fleiss, Tytun, Ury
ftuss	Sample size for 2-sample binomial test using " " " " (Both by Dan Heitjan, <dheitjan@biostats.hmc.psu.edu>)
gbayes	Bayesian posterior and predictive distributions when both the prior and the likelihood are Gaussian
getHdata	Fetch and list datasets on our web site
hdquantile	Harrell-Davis nonparametric quantile estimator with s.e.
histbackback	Back-to-back histograms (Pat Burns, Salomon Smith Barney, London, <pburns@dorado.sbi.com>)
hist.data.frame	Matrix of histograms for all numeric vars. in data frame Use hist.data.frame(data.frame.name)
histSpike	Add high-resolution spike histograms or density estimates to an existing plot
hoeffd	Hoeffding's D test (omnibus test of independence of X and Y)
impute	Impute missing data (generic method)
interaction	More flexible version of builtin function
is.present	Tests for non-blank character values or non-NA numeric values
james.stein	James-Stein shrinkage estimates of cell means from raw data
labcurve	Optimally label a set of curves that have been drawn on an existing plot, on the basis of gaps between curves. Also position legends automatically at emptiest rectangle.
label	Set or fetch a label for an R-object
Lag	Lag a vector, padding on the left with NA or ""
latex	Convert an R object to LaTeX (R Heiberger & FE Harrell)
list.tree	Pretty-print the structure of any data object (Alan Zaslavsky, <zaslavsk@hcp.med.harvard.edu>)
Load	Enhancement of load
mask	8-bit logical representation of a short integer value (Rick Becker)
matchCases	Match each case on one continuous variable
matxv	Fast matrix * vector, handling intercept(s) and NAs
mgp.axis	Version of axis() that uses appropriate mgp from mgp.axis.labels and gets around bug in axis(2, ...) that causes it to assume las=1
mgp.axis.labels	Used by survplot and plot in <b>rms</b> library (and other functions in the future) so that different spacing between tick marks and axis tick mark labels may be

	specified for x- and y-axes. Use <code>mgp.axis.labels('default')</code> to set defaults. Users can set values manually using <code>mgp.axis.labels(x,y)</code> where x and y are 2nd value of <code>par('mgp')</code> to use. Use <code>mgp.axis.labels(type=w)</code> to retrieve values, where w='x', 'y', 'x and y', 'xy', to get 3 mgp values (first 3 types) or 2 <code>mgp.axis.labels</code> .
<code>minor.tick</code>	Add minor tick marks to an existing plot
<code>mtitle</code>	Add outer titles and subtitles to a multiple plot layout
<code>multLines</code>	Draw multiple vertical lines at each x in a line plot
<code>%nin%</code>	Opposite of <code>%in%</code>
<code>nobsY</code>	Compute no. non-NA observations for left hand formula side
<code>nomiss</code>	Return a matrix after excluding any row with an NA
<code>panel.bplot</code>	Panel function for trellis <code>bplot</code> - box-percentile plots
<code>panel.plsmo</code>	Panel function for trellis <code>xyplot</code> - uses <code>plsmo</code>
<code>pBlock</code>	Block variables for certain lattice charts
<code>pc1</code>	Compute first prin. component and get coefficients on original scale of variables
<code>plotCorrPrecision</code>	Plot precision of estimate of correlation coefficient
<code>plsmo</code>	Plot smoothed x vs. y with labeling and exclusion of NAs Also allows a grouping variable and plots unsmoothed data
<code>popower</code>	Power and sample size calculations for ordinal responses (two treatments, proportional odds model)
<code>prn</code>	<code>prn(expression)</code> does <code>print(expression)</code> but titles the output with 'expression'. Do <code>prn(expression,txt)</code> to add a heading ('txt') before the 'expression' title
<code>pstamp</code>	Stamp a plot with date in lower right corner ( <code>pstamp()</code> ) Add <code>,pwd=T</code> and/or <code>,time=T</code> to add current directory name or time Put additional text for label as first argument, e.g. <code>pstamp('Figure 1')</code> will draw 'Figure 1 date'
<code>putKey</code>	Different way to use <code>key()</code>
<code>putKeyEmpty</code>	Put key at most empty part of existing plot
<code>rcorr</code>	Pearson or Spearman correlation matrix with pairwise deletion of missing data
<code>rcorr.cens</code>	Somers' $D_{xy}$ rank correlation with censored data
<code>rcorrr.cens</code>	Assess difference in concordance for paired predictors
<code>rcspline.eval</code>	Evaluate restricted cubic spline design matrix
<code>rcspline.plot</code>	Plot spline fit with nonparametric smooth and grouped estimates
<code>rcspline.restate</code>	Restate restricted cubic spline in unrestricted form, and create TeX expression to print the fitted function
<code>reShape</code>	Reshape a matrix into 3 vectors, reshape serial data
<code>rm.boot</code>	Bootstrap spline fit to repeated measurements model, with simultaneous confidence region - least squares using spline function in time
<code>rMultinom</code>	Generate multinomial random variables with varying prob.
<code>samplesize.bin</code>	Sample size for 2-sample binomial problem

	(Rick Chappell, <chappell@stat.wisc.edu>)
sas.get	Convert SAS dataset to S data frame
sasxport.get	Enhanced importing of SAS transport dataset in R
Save	Enhancement of save
scat1d	Add 1-dimensional scatterplot to an axis of an existing plot (like bar-codes, FEH/Martin Maechler, <maechler@stat.math.ethz.ch>/Jens Oehlschlaegel-Akiyoshi, <oehl@psyres-stuttgart.de>)
score.binary	Construct a score from a series of binary variables or expressions
sedit	A set of character handling functions written entirely in R. sedit() does much of what the UNIX sed program does. Other functions included are substring.location, substring<-, replace.string.wild, and functions to check if a string is numeric or contains only the digits 0-9
setTrellis	Set Trellis graphics to use blank conditioning panel strips, line thickness 1 for dot plot reference lines: setTrellis(); 3 optional arguments
show.col	Show colors corresponding to col=0,1,...,99
show.pch	Show all plotting characters specified by pch=. Just type show.pch() to draw the table on the current device.
showPsfrag	Use LaTeX to compile, and dvips and ghostview to display a postscript graphic containing psfrag strings
solvet	Version of solve with argument tol passed to qr
somers2	Somers' rank correlation and c-index for binary y
spearman	Spearman rank correlation coefficient spearman(x,y)
spearman.test	Spearman 1 d.f. and 2 d.f. rank correlation test
spearman2	Spearman multiple d.f. $\rho^2$ , adjusted $\rho^2$ , Wilcoxon-Kruskal- Wallis test, for multiple predictors
spower	Simulate power of 2-sample test for survival under complex conditions Also contains the Gompertz2, Weibull2, Lognorm2 functions.
spss.get	Enhanced importing of SPSS files using read.spss function
src	src(name) = source("name.s") with memory
store	store an object permanently (easy interface to assign function)
strmatch	Shortest unique identifier match (Terry Therneau, <therneau@mayo.edu>)
subset	More easily subset a data frame
substi	Substitute one var for another when observations NA
summarize	Generate a data frame containing stratified summary statistics. Useful for passing to trellis.
summary.formula	General table making and plotting functions for summarizing data
summaryD	Summarizing using user-provided formula and dotchart3
summaryM	Replacement for summary.formula(..., method='reverse')
summaryP	Multi-panel dot chart for summarizing proportions

summaryS	Summarize multiple response variables for multi-panel dot chart or scatterplot
summaryRc	Summary for continuous variables using lowess
symbol.freq	X-Y Frequency plot with circles' area prop. to frequency
sys	Execute unix() or dos() depending on what's running
tabulr	Front-end to tabular function in the tables package
tex	Enclose a string with the correct syntax for using with the LaTeX psfrag package, for postscript graphics
transace	ace() packaged for easily automatically transforming all variables in a matrix
transcan	automatic transformation and imputation of NAs for a series of predictor variables
trap.rule	Area under curve defined by arbitrary x and y vectors, using trapezoidal rule
trellis.strip.blank	To make the strip titles in trellis more visible, you can make the backgrounds blank by saying trellis.strip.blank(). Use before opening the graphics device.
t.test.cluster	2-sample t-test for cluster-randomized observations
uncbind	Form individual variables from a matrix
upData	Update a data frame (change names, labels, remove vars, etc.)
units	Set or fetch "units" attribute - units of measurement for var.
varclus	Graph hierarchical clustering of variables using squared Pearson or Spearman correlations or Hoeffding D as similarities Also includes the naclus function for examining similarities in patterns of missing values across variables.
wtd.mean	
wtd.var	
wtd.quantile	
wtd.Ecdf	
wtd.table	
wtd.rank	
wtd.loess.noiter	
num.denom.setup	Set of function for obtaining weighted estimates
xy.group	Compute mean x vs. function of y by groups of x
xYplot	Like trellis xyplot but supports error bars and multiple response variables that are connected as separate lines
ynbind	Combine a series of yes/no true/false present/absent variables into a matrix
zoom	Zoom in on any graphical display (Bill Dunlap, <bill@statsci.com>)

### Copyright Notice

#### GENERAL DISCLAIMER

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

In short: You may use it any way you like, as long as you don't charge money for it, remove this notice, or hold anyone liable for its results. Also, please acknowledge the source and communicate changes to the author.

If this software is used in work presented for publication, kindly reference it using for example: Harrell FE (2014): Hmisc: A package of miscellaneous R functions. Programs available from <https://hbiostat.org/R/Hmisc/>.

Be sure to reference R itself and other libraries used.

### Author(s)

Frank E Harrell Jr  
Professor of Biostatistics  
Vanderbilt University School of Medicine  
Nashville, Tennessee  
<fh@fharrell.com>

### References

See Alzola CF, Harrell FE (2004): An Introduction to S and the Hmisc and Design Libraries at <https://hbiostat.org/R/doc/sintro.pdf> for extensive documentation and examples for the Hmisc package.

---

hoeffd

*Matrix of Hoeffding's D Statistics*

---

### Description

Computes a matrix of Hoeffding's (1948) D statistics for all possible pairs of columns of a matrix. D is a measure of the distance between  $F(x, y)$  and  $G(x)H(y)$ , where  $F(x, y)$  is the joint CDF of X and Y, and G and H are marginal CDFs. Missing values are deleted in pairs rather than deleting all rows of x having any missing variables. The D statistic is robust against a wide variety of alternatives to independence, such as non-monotonic relationships. The larger the value of D, the more dependent are X and Y (for many types of dependencies). D used here is 30 times Hoeffding's original D, and ranges from -0.5 to 1.0 if there are no ties in the data. `print.hoeffd` prints the information derived by `hoeffd`. The higher the value of D, the more dependent are x and y. `hoeffd` also computes the mean and maximum absolute values of the difference between the joint empirical CDF and the product of the marginal empirical CDFs.

**Usage**

```
hoeffd(x, y)
## S3 method for class 'hoeffd'
print(x, ...)
```

**Arguments**

x	a numeric matrix with at least 5 rows and at least 2 columns (if y is absent), or an object created by hoeffd
y	a numeric vector or matrix which will be concatenated to x
...	ignored

**Details**

Uses midranks in case of ties, as described by Hollander and Wolfe. P-values are approximated by linear interpolation on the table in Hollander and Wolfe, which uses the asymptotically equivalent Blum-Kiefer-Rosenblatt statistic. For  $P < .0001$  or  $> 0.5$ , P values are computed using a well-fitting linear regression function in  $\log P$  vs. the test statistic. Ranks (but not bivariate ranks) are computed using efficient algorithms (see reference 3).

**Value**

a list with elements D, the matrix of D statistics, n the matrix of number of observations used in analyzing each pair of variables, and P, the asymptotic P-values. Pairs with fewer than 5 non-missing values have the D statistic set to NA. The diagonals of n are the number of non-NAs for the single variable corresponding to that row and column.

**Author(s)**

Frank Harrell  
Department of Biostatistics  
Vanderbilt University  
<fh@fharrell.com>

**References**

- Hoeffding W. (1948): A non-parametric test of independence. *Ann Math Stat* 19:546–57.
- Hollander M. and Wolfe D.A. (1973). *Nonparametric Statistical Methods*, pp. 228–235, 423. New York: Wiley.
- Press WH, Flannery BP, Teukolsky SA, Vetterling, WT (1988): *Numerical Recipes in C*. Cambridge: Cambridge University Press.

**See Also**

[rcorr](#), [varclus](#)



**Examples**

```
x <- c(-2, -1, 0, 1, 2)
y <- c(4, 1, 0, 1, 4)
z <- c(1, 2, 3, 4, NA)
q <- c(1, 2, 3, 4, 5)
hoeffd(cbind(x,y,z,q))

# Hoeffding's test can detect even one-to-many dependency
set.seed(1)
x <- seq(-10,10,length=200)
y <- x*sign(runif(200,-1,1))
plot(x,y)
hoeffd(x,y)
```

---

html

---

*Convert an S object to HTML*


---

**Description**

html is a generic function, for which only two methods are currently implemented, `html.latex` and a rudimentary `html.data.frame`. The former uses the HeVeA LaTeX to HTML translator by Maranget to create an HTML file from a LaTeX file like the one produced by latex. `html.default` just runs `html.data.frame`. `htmlVerbatim` prints all of its arguments to the console in an html verbatim environment, using a specified percent of the prevailing character size. This is useful for R Markdown with knitr.

Most of the html-producing functions in the Hmisc and rms packages return a character vector passed through `htmltools::HTML` so that knitr will correctly format the result without the need for the user putting `results='asis'` in the chunk header.

**Usage**

```
html(object, ...)
## S3 method for class 'latex'
html(object, file, where=c('cwd', 'tmp'),
      method=c('hevea', 'htlatex'),
      rmarkdown=FALSE, cleanup=TRUE, ...)
## S3 method for class 'data.frame'
html(object,
      file=paste(first.word(deparse(substitute(object))), 'html', sep='.'), header,
      caption=NULL, rownames=FALSE, align='r', align.header='c',
      bold.header=TRUE, col.header='Black',
      border=2, width=NULL, size=100, translate=FALSE,
      append=FALSE, link=NULL, linkCol=1,
      linkType=c('href', 'name'), disableq=FALSE, ...)
## Default S3 method:
html(object,
```

```

file=paste(first.word(deparse(substitute(object))), 'html', sep='.'),
append=FALSE, link=NULL, linkCol=1, linkType=c('href', 'name'), ...)
htmlVerbatim(..., size=75, width=85, scroll=FALSE, rows=10, cols=100,
propts=NULL, omit1b=FALSE)

```

### Arguments

object	a data frame or an object created by latex. For the generic html is any object for which an html method exists.
file	name of the file to create. The default file name is object.html where object is the first word in the name of the argument for object. For html.latex specify file='' or file=character(0) to print html code to the console, as when using knitr. For the data.frame method, file may be set to FALSE which causes a character vector enclosed in htmltools::HTML to be returned instead of writing to the console.
where	for html. Default is to put output files in current working directory. Specify where='tmp' to put in a system temporary directory area.
method	default is to use system command hevea to convert from LaTeX to html. Specify method='htlatex' to use system command htlatex, assuming the system package TeX4ht is installed.
rmarkdown	set to TRUE if using RMarkdown (usually under knitr and RStudio). This causes html to be packaged for RMarkdown and output to go into the console stream. file is ignored when rmarkdown=TRUE.
cleanup	if using method='htlatex' set to FALSE if where='cwd' to prevent deletion of auxiliary files created by htlatex that are not needed when using the final html document (only the .css file is needed in addition to .html). If using method='hevea', cleanup=TRUE causes deletion of the generated .haux file.
header	vector of column names. Defaults to names in object. Set to NULL to suppress column names.
caption	a character string to be used as a caption before the table
rownames	set to FALSE to ignore row names even if they are present
align	alignment for table columns (all are assumed to have the same if is a scalar). Specify "c", "r", "l" for center, right, or left alignment.
align.header	same coding as for align but pertains to header
bold.header	set to FALSE to not bold face column headers
col.header	color for column headers
border	set to 0 to not include table cell borders, 1 to include only outer borders, or 2 (the default) to put borders around cells too
translate	set to TRUE to run header and table cell text through the htmlTranslate function
width	optional table width for html.data.frame. For full page width use width="100%", for use in options() for printing objects.
size	a number between 0 and 100 representing the percent of the prevailing character size to be used by htmlVerbatim and the data frame method.
append	set to TRUE to append to an existing file

link	character vector specifying hyperlink names to attach to selected elements of the matrix or data frame. No hyperlinks are used if link is omitted or for elements of link that are "". To allow multiple links per link, link may also be a character matrix shaped as object in which case linkCol is ignored.
linkCol	column number of object to which hyperlinks are attached. Defaults to first column.
linkType	defaults to "href"
disableq	set to TRUE to add code to the html table tag that makes Quarto not use its usual table style
...	ignored except for htmlVerbatim - is a list of objects to print()
scroll	set to TRUE to put the html in a scrollable textarea
rows, cols	the number of rows and columns to devote to the visible part of the scrollable box
propts	options, besides quote=FALSE to pass to the print method, for htmlVerbatim
omit1b	for htmlVerbatim if TRUE causes an initial and a final line of output that is all blank to be deleted

**Author(s)**

Frank E. Harrell, Jr.  
 Department of Biostatistics,  
 Vanderbilt University,  
 <fh@fharrell.com>

**References**

Maranget, Luc. HeVeA: a LaTeX to HTML translator. URL: <http://para.inria.fr/~maranget/hevea/>

**See Also**

[latex](#)

**Examples**

```
## Not run:
x <- matrix(1:6, nrow=2, dimnames=list(c('a','b'),c('c','d','e')))
w <- latex(x)
h <- html(w) # run HeVeA to convert .tex to .html
h <- html(x) # convert x directly to html
w <- html(x, link=c('','B')) # hyperlink first row first col to B

# Assuming system package tex4ht is installed, easily convert advanced
# LaTeX tables to html
getHdata(pbc)
s <- summaryM(bili + albumin + stage + protime + sex + age + spiders ~ drug,
              data=pbc, test=TRUE)
w <- latex(s, npct='slash', file='s.tex')
z <- html(w)
```

```
browseURL(z$file)

d <- describe(pbc)
w <- latex(d, file='d.tex')
z <- html(w)
browseURL(z$file)

## End(Not run)
```

---

htmltabv

*htmltabc*

---

## Description

Simple HTML Table of Verbatim Output

## Usage

```
htmltabv(..., cols = 2, propts = list(quote = FALSE))
```

## Arguments

...	objects to <code>print()</code> . The arguments must be named with the labels you want to print before the verbatim <code>print()</code> .
cols	number of columns in the html table
propts	an option list of arguments to pass to the <code>print()</code> methods; default is to not quote character strings

## Details

Uses `capture.output` to capture as character strings the results of running `print()` on each element of `...`. If an element of `...` has length of 1 and is a blank string, nothing is printed for that cell other than its name (not in verbatim).

## Value

character string of html

## Author(s)

Frank Harrell

**Description**

These functions do simple and transcan imputation and print, summarize, and subscript variables that have NAs filled-in with imputed values. The simple imputation method involves filling in NAs with constants, with a specified single-valued function of the non-NAs, or from a sample (with replacement) from the non-NA values (this is useful in multiple imputation). More complex imputations can be done with the transcan function, which also works with the generic methods shown here, i.e., impute can take a transcan object and use the imputed values created by transcan (with imputed=TRUE) to fill-in NAs. The print method places \* after variable values that were imputed. The summary method summarizes all imputed values and then uses the next summary method available for the variable. The subscript method preserves attributes of the variable and subsets the list of imputed values corresponding with how the variable was subsetted. The is.imputed function is for checking if observations are imputed.

**Usage**

```
impute(x, ...)

## Default S3 method:
impute(x, fun=median, ...)

## S3 method for class 'impute'
print(x, ...)

## S3 method for class 'impute'
summary(object, ...)

is.imputed(x)
```

**Arguments**

x	a vector or an object created by transcan, or a vector needing basic unconditional imputation. If there are no NAs and x is a vector, it is returned unchanged.
fun	the name of a function to use in computing the (single) imputed value from the non-NAs. The default is median. If instead of specifying a function as fun, a single value or vector (numeric, or character if object is a factor) is specified, those values are used for insertion. fun can also be the character string "random" to draw random values for imputation, with the random values not forced to be the same if there are multiple NAs. For a vector of constants, the vector must be of length one (indicating the same value replaces all NAs) or must be as long as the number of NAs, in which case the values correspond to consecutive NAs to replace. For a factor object, constants for imputation may include character values not in the current levels of object. In that case new levels are added. If

object is of class "factor", fun is ignored and the most frequent category is used for imputation.

object	an object of class "impute"
...	ignored

**Value**

a vector with class "impute" placed in front of existing classes. For `is.imputed`, a vector of logical values is returned (all TRUE if object is not of class impute).

**Author(s)**

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>

**See Also**

[transcan](#), [impute.transcan](#), [describe](#), [na.include](#), [sample](#)

**Examples**

```
age <- c(1,2,NA,4)
age.i <- impute(age)
# Could have used impute(age,2.5), impute(age,mean), impute(age,"random")
age.i
summary(age.i)
is.imputed(age.i)
```

---

intMarkovOrd

*intMarkovOrd*


---

**Description**

Compute Parameters for Proportional Odds Markov Model

**Usage**

```
intMarkovOrd(
  y,
  times,
  initial,
  absorb = NULL,
  intercepts,
  extra = NULL,
  g,
  target,
```

```

    t,
    ftarget = NULL,
    onlycrit = FALSE,
    constraints = NULL,
    printsop = FALSE,
    ...
)

```

### Arguments

<code>y</code>	vector of possible <code>y</code> values in order (numeric, character, factor)
<code>times</code>	vector of measurement times
<code>initial</code>	initial value of <code>y</code> (baseline state; numeric, character, or factor matching <code>y</code> ). If length 1 this value is used for all subjects, otherwise it is a vector of length <code>n</code> .
<code>absorb</code>	vector of absorbing states, a subset of <code>y</code> (numeric, character, or factor matching <code>y</code> ). The default is no absorbing states. Observations are truncated when an absorbing state is simulated.
<code>intercepts</code>	vector of initial guesses for the intercepts
<code>extra</code>	an optional vector of initial guesses for other parameters passed to <code>g</code> such as regression coefficients for previous states and for general time trends. Name the elements of <code>extra</code> for more informative output.
<code>g</code>	a user-specified function of three or more arguments which in order are <code>yprev</code> - the value of <code>y</code> at the previous time, the current time <code>t</code> , the gap between the previous time and the current time, an optional (usually named) covariate vector <code>X</code> , and optional arguments such as a regression coefficient value to simulate from. The function needs to allow <code>yprev</code> to be a vector and <code>yprev</code> must not include any absorbing states. The <code>g</code> function returns the linear predictor for the proportional odds model aside from <code>intercepts</code> . The returned value must be a matrix with row names taken from <code>yprev</code> . If the model is a proportional odds model, the returned value must be one column. If it is a partial proportional odds model, the value must have one column for each distinct value of the response variable <code>Y</code> after the first one, with the levels of <code>Y</code> used as optional column names. So columns correspond to <code>intercepts</code> . The different columns are used for <code>y</code> -specific contributions to the linear predictor (aside from <code>intercepts</code> ) for a partial or constrained partial proportional odds model. Parameters for partial proportional odds effects may be included in the ... arguments.
<code>target</code>	vector of target state occupancy probabilities at time <code>t</code> . If <code>extra</code> is specified, <code>target</code> must be a matrix where row names are character versions of <code>t</code> and columns represent occupancy probabilities corresponding to values of <code>y</code> at the time given in the row.
<code>t</code>	target times. Can have more than one element only if <code>extra</code> is given.
<code>ftarget</code>	an optional function defining constraints that relate to transition probabilities. The function returns a penalty which is a sum of absolute differences in probabilities from target probabilities over possibly multiple targets. The <code>ftarget</code> function must have two arguments: <code>intercepts</code> and <code>extra</code> .
<code>onlycrit</code>	set to TRUE to only return the achieved objective criterion and not print anything

constraints	a function of two arguments: the vector of current intercept values and the vector of extra parameters, returning TRUE if that vector meets the constraints and FALSE otherwise
printsop	set to TRUE to print solved-for state occupancy probabilities for groups 1 and 2 and log odds ratios corresponding to them
...	optional arguments to pass to <code>stats::nlm()</code> . If this is specified, the arguments that <code>intMarkovOrd</code> normally sends to <code>nlm</code> are not used.

### Details

Given a vector `intercepts` of initial guesses at the intercepts in a Markov proportional odds model, and a vector `extra` if there are other parameters, solves for the `intercepts` and `extra` vectors that yields a set of occupancy probabilities at time `t` that equal, as closely as possible, a vector of target values.

### Value

list containing two vectors named `intercepts` and `extra` unless `oncrit=TRUE` in which case the best achieved sum of absolute errors is returned

### Author(s)

Frank Harrell

### See Also

<https://hbiostat.org/R/Hmisc/markov/>

---

knitrSet

*knitr Setup and plotly Service Function*

---

### Description

`knitrSet` sets up `knitr` to use better default parameters for base graphics, better code formatting, and to allow several arguments to be passed from code chunk headers, such as `bty`, `mfrow`, `ps`, `bot` (extra bottom margin for base graphics), `top` (extra top margin), `left` (extra left margin), `rt` (extra right margin), `lwd`, `mgp`, `las`, `tcl`, `axes`, `xpd`, `h` (usually `fig.height` in `knitr`), `w` (usually `fig.width` in `knitr`), `wo` (`out.width` in `knitr`), `ho` (`out.height` in `knitr`), `cap` (character string containing figure caption), `scap` (character string containing short figure caption for table of figures). The `capfile` argument facilitates auto-generating a table of figures for certain Rmarkdown report themes. This is done by the addition of a hook function that appends data to the `capfile` file each time a chunk runs that has a long or short caption in the chunk header.

`plotlySave` saves a `plotly` graphic with name `foo.png` where `foo` is the name of the current chunk. You must have a free `plotly` account from `plot.ly` to use this function, and you must have run `Sys.setenv(plotly_username="your_plotly_username")` and `Sys.setenv(plotly_api_key="your_api_key")`. The API key can be found in one's profile settings.



**Usage**

```
knitrSet(basename=NULL, w=if(! bd) 4, h=if(! bd) 3, wo=NULL, ho=NULL,
        fig.path=if(length(basename)) basename else '',
        fig.align=if(! bd) 'center', fig.show='hold',
        fig.pos=if(! bd) 'htbp',
        fig.lp = if(! bd) paste('fig', basename, sep=':'),
        dev=switch(lang, latex='pdf', markdown='png',
                  blogdown=NULL, quarto=NULL),
        tidy=FALSE, error=FALSE,
        messages=c('messages.txt', 'console'),
        width=61, decinline=5, size=NULL, cache=FALSE,
        echo=TRUE, results='markup', capfile=NULL,
        lang=c('latex', 'markdown', 'blogdown', 'quarto'))
```

```
plotlySave(x, ...)
```

**Arguments**

basename	base name to be added in front of graphics file names. basename is followed by a minus sign.
w, h	default figure width and height in inches
wo, ho	default figure rendering width and height, in integer pixels or percent as a character string, e.g. '40%'
fig.path	path for figures. To put figures in a subdirectory specify e.g. fig.path='folder/'. Ignored for blogdown.
fig.align, fig.show, fig.pos, fig.lp, tidy, cache, echo, results, error, size	see knitr documentation
dev	graphics device, with default figured from lang
messages	By default warning and other messages such as those from loading packages are sent to file 'messages.txt' in the current working directory. You can specify messages='console' to send them directly to the console.
width	text output width for R code and output
decinline	number of digits to the right of the decimal point to round numeric values appearing inside Sexpr
capfile	the name of a file in the current working directory that is used to accumulate chunk labels, figure cross-reference tags, and figure short captions (long captions if no short caption is defined) for the purpose of using <code>markupSpecs\$markdown\$tof()</code> to insert a table of figures in a report. The file is appended to, which is useful if cache=TRUE is used since this will keep some chunks from running. The tof function will remove earlier duplicated figure tags if this is the case. If not caching, the user should initialize the file to empty at the top of the script.
lang	Default is 'latex' to use LaTeX. Set to 'markdown' when using R Markdown or 'blogdown' or 'quarto'. For 'blogdown' and 'quarto', par and knitr graphics-related hooks are not called as this would prevent writing graphics files in the correct directory for the blog system.

`x` a plotly graphics object or a named list of such objects. The resulting png file will go in the file path given by the knitr `fig.path` value, and have a base name equal to the current knitr chunk name. If `x` is a list, a minus sign followed by the chunk name are inserted before `.png`.

`...` additional arguments passed to `plotly::plotly_IMAGE`

**Author(s)**

Frank Harrell

**See Also**

[knit](#)

**Examples**

```
## Not run:
# Typical call (without # comment symbols):
# <<echo=FALSE>>=
# require(Hmisc)
# knitrSet()
# @

knitrSet() # use all defaults and don't use a graphics file prefix
knitrSet('modeling') # use modeling- prefix for a major section or chapter
knitrSet(cache=TRUE, echo=FALSE) # global default to cache and not print code
knitrSet(w=5,h=3.75) # override default figure width, height

# ```{r chunkname}
# p <- plotly::plot_ly(...)
# plotlySave(p) # creates fig.path/chunkname.png

## End(Not run)
```

---

labcurve

*Label Curves, Make Keys, and Interactively Draw Points and Curves*

---

**Description**

`labcurve` optionally draws a set of curves then labels the curves. A variety of methods for drawing labels are implemented, ranging from positioning using the mouse to automatic labeling to automatic placement of key symbols with manual placement of key legends to automatic placement of legends. For automatic positioning of labels or keys, a curve is labeled at a point that is maximally separated from all of the other curves. Gaps occurring when curves do not start or end at the same x-coordinates are given preference for positioning labels. If labels are offset from the curves (the default behaviour), if the closest curve to curve `i` is above curve `i`, curve `i` is labeled below its line. If the closest curve is below curve `i`, curve `i` is labeled above its line. These directions are reversed if the resulting labels would appear outside the plot region.

Both ordinary lines and step functions are handled, and there is an option to draw the labels at the same angle as the curve within a local window.

Unless the mouse is used to position labels or plotting symbols are placed along the curves to distinguish them, curves are examined at 100 (by default) equally spaced points over the range of x-coordinates in the current plot area. Linear interpolation is used to get y-coordinates to line up (step function or constant interpolation is used for step functions). There is an option to instead examine all curves at the set of unique x-coordinates found by unioning the x-coordinates of all the curves. This option is especially useful when plotting step functions. By setting `adj="auto"` you can have `labcurve` try to optimally left- or right-justify labels depending on the slope of the curves at the points at which labels would be centered (plus a vertical offset). This is especially useful when labels must be placed on steep curve sections.

You can use the `on top` method to write (short) curve names directly on the curves (centered on the y-coordinate). This is especially useful when there are many curves whose full labels would run into each other. You can plot letters or numbers on the curves, for example (using the `keys` option), and have `labcurve` use the `key` function to provide long labels for these short ones (see the end of the example). There is another option for connecting labels to curves using arrows. When `keys` is a vector of integers, it is taken to represent plotting symbols (`pchs`), and these symbols are plotted at equally-spaced x-coordinates on each curve (by default, using 5 points per curve). The points are offset in the x-direction between curves so as to minimize the chance of collisions.

To add a legend defining line types, colors, or line widths with no symbols, specify `keys="lines"`, e.g., `labcurve(curves,keys="lines", lty=1:2)`.

`putKey` provides a different way to use `key()` by allowing the user to specify vectors for labels, line types, plotting characters, etc. Elements that do not apply (e.g., `pch` for lines (`type="l"`)) may be `NA`. When a series of points is represented by both a symbol and a line, the corresponding elements of both `pch` and `lty`, `col`, or `lwd` will be non-missing.

`putKeyEmpty`, given vectors of all the x-y coordinates that have been plotted, uses `largest.empty` to find the largest empty rectangle large enough to hold the key, and draws the key using `putKey`.

`drawPlot` is a simple mouse-driven function for drawing series of lines, step functions, polynomials, Bezier curves, and points, and automatically labeling the point groups using `labcurve` or `putKeyEmpty`. When `drawPlot` is invoked it creates temporary functions `Points`, `Curve`, and `Abline`. The user calls these functions inside the call to `drawPlot` to define groups of points in the order they are defined with the mouse. `Abline` is used to call `abline` and not actually great a group of points. For some curve types, the curve generated to represent the corresponding series of points is drawn after all points are entered for that series, and this curve may be different than the simple curve obtained by connecting points at the mouse clicks. For example, to draw a general smooth Bezier curve the user need only click on a few points, and she must overshoot the final curve coordinates to define the curve. The originally entered points are not erased once the curve is drawn. The same goes for step functions and polynomials. If you `plot()` the object returned by `drawPlot`, however, only final curves will be shown. The last examples show how to use `drawPlot`.

The `largest.empty` function finds the largest rectangle that is large enough to hold a rectangle of a given height and width, such that the rectangle does not contain any of a given set of points. This is used by `labcurve` and `putKeyEmpty` to position keys at the most empty part of an existing plot. The default method was created by Hans Borchers.

## Usage

```
labcurve(curves, labels=names(curves),
```

```

method=NULL, keys=NULL, keyloc=c("auto", "none"),
type="l", step.type=c("left", "right"),
xmethod=if(any(type=="s")) "unique" else "grid",
offset=NULL, xlim=NULL,
tilt=FALSE, window=NULL, npts=100, cex=NULL,
adj="auto", angle.adj.auto=30,
lty=pr$lty, lwd=pr$lwd, col.=pr$col, transparent=TRUE,
arrow.factor=1, point.inc=NULL, opts=NULL, key.opts=NULL,
empty.method=c('area', 'maxdim'), numbins=25,
pl=!missing(add), add=FALSE,
ylim=NULL, xlab="", ylab="",
whichLabel=1:length(queries),
grid=FALSE, xrestrict=NULL, ...)

putKey(z, labels, type, pch, lty, lwd,
      cex=par('cex'), col=rep(par('col'),nc),
      transparent=TRUE, plot=TRUE, key.opts=NULL, grid=FALSE)

putKeyEmpty(x, y, labels, type=NULL,
           pch=NULL, lty=NULL, lwd=NULL,
           cex=par('cex'), col=rep(par('col'),nc),
           transparent=TRUE, plot=TRUE, key.opts=NULL,
           empty.method=c('area', 'maxdim'),
           numbins=25,
           xlim=pr$usr[1:2], ylim=pr$usr[3:4], grid=FALSE)

drawPlot(..., xlim=c(0,1), ylim=c(0,1), xlab='', ylab='',
         ticks=c('none', 'x', 'y', 'xy'),
         key=FALSE, opts=NULL)

# Points(label=' ', type=c('p', 'r'),
#        n, pch=pch.to.use[1], cex=par('cex'), col=par('col'),
#        rug = c('none', 'x', 'y', 'xy'), ymean)

# Curve(label=' ',
#        type=c('bezier', 'polygon', 'linear', 'pol', 'loess', 'step', 'gauss'),
#        n=NULL, lty=1, lwd=par('lwd'), col=par('col'), degree=2,
#        evaluation=100, ask=FALSE)

# Abline(\dots)

## S3 method for class 'drawPlot'
plot(x, xlab, ylab, ticks,
     key=x$key, keyloc=x$keyloc, ...)

largest.empty(x, y, width=0, height=0,
             numbins=25, method=c('exhaustive', 'rexhaustive', 'area', 'maxdim'),
             xlim=pr$usr[1:2], ylim=pr$usr[3:4],

```

pl=FALSE, grid=FALSE)

### Arguments

curves	a list of lists, each of which have at least two components: a vector of x values and a vector of corresponding y values. curves is mandatory except when method="mouse" or "locator", in which case labels is mandatory. Each list in curves may optionally have any of the parameters type, lty, lwd, or col for that curve, as defined below (see one of the last examples).
z	a two-element list specifying the coordinate of the center of the key, e.g. locator(1) to use the mouse for positioning
labels	For labcurve, a vector of character strings used to label curves (which may contain newline characters to stack labels vertically). The default labels are taken from the names of the curves list. Setting labels=FALSE will suppress drawing any labels (for labcurve only). For putKey and putKeyEmpty is a vector of character strings specifying group labels
x	see below
y	for putKeyEmpty and largest.empty, x and y are same-length vectors specifying points that have been plotted. x can also be an object created by drawPlot.
...	For drawPlot is a series of invocations of Points and Curve (see example). Any number of point groups can be defined in this way. For Abline these may be any arguments to abline. For labcurve, other parameters to pass to text.
width	see below
height	for largest.empty, specifies the minimum allowable width in x units and the minimum allowable height in y units
method	"offset" (the default) offsets labels at largest gaps between curves, and draws labels beside curves. "on top" draws labels on top of the curves (especially good when using keys). "arrow" draws arrows connecting labels to the curves. "mouse" or "locator" positions labels according to mouse clicks. If keys is specified and is an integer vector or is "lines", method defaults to "on top". If keys is character, method defaults to "offset". Set method="none" to suppress all curve labeling and key drawing, which is useful when pl=TRUE and you only need labcurve to draw the curves and the rest of the basic graph.  For largest.empty specifies the method a rectangle that does not collide with any of the (x, y) points. The default method, 'exhaustive', uses a Fortran translation of an R function and algorithm developed by Hans Borchers. The same result, more slowly, may be obtained by using pure R code by specifying method='rexhaustive'. The original algorithms using binning (and the only methods supported for S-Plus) are still available. For all methods, screening of candidate rectangles having at least a given width in x-units of width or having at least a given height in y-units of height is possible. Use method="area" to use the binning method to find the rectangle having the largest area, or method="maxdim" to use the binning method to return with last rectangle searched that had both the largest width and largest height over all previous rectangles.

keys	This causes keys (symbols or short text) to be drawn on or beside curves, and if keyloc is not equal to "none", a legend to be automatically drawn. The legend links keys with full curve labels and optionally with colors and line types. Set keys to a vector of character strings, or a vector of integers specifying plotting character (pch values - see points). For the latter case, the default behavior is to plot the symbols periodically, at equally spaced x-coordinates.
keyloc	When keys is specified, keyloc specifies how the legend is to be positioned for drawing using the key function in trellis. The default is "auto", for which the largest.empty function is used to find the most empty part of the plot. If no empty rectangle large enough to hold the key is found, no key will be drawn. Specify keyloc="none" to suppress drawing a legend, or set keyloc to a 2-element list containing the x and y coordinates for the center of the legend. For example, use keyloc=locator(1) to click the mouse at the center. keyloc specifies the coordinates of the center of the key to be drawn with plot.drawPlot when key=TRUE.
type	for labcurve, a scalar or vector of character strings specifying the method that the points in the curves were connected. "l" means ordinary connections between points and "s" means step functions. For putKey and putKeyEmpty is a vector of plotting types, "l" for regular line, "p" for point, "b" for both point and line, and "n" for none. For Points is either "p" (the default) for regular points, or "r" for rugplot (one-dimensional scatter diagram to be drawn using the scat1d function). For Curve, type is "bezier" (the default) for drawing a smooth Bezier curves (which can represent a non-1-to-1 function such as a circle), "polygon" for ordinary line segments, "linear" for a straight line defined by two endpoints, "pol" for a degree-degree polynomial to be fitted to the mouse-clicked points, "step" for a left-step-function, "gauss" to plot a Gaussian density fitted to 3 clicked points, "loess" to use the lowess function to smooth the clicked points, or a function to draw a user-specified function, evaluated at evaluation points spanning the whole x-axis. For the density the user must click in the left tail, at the highest value (at the mean), and in the right tail, with the two tail values being approximately equidistant from the mean. The density is scaled to fit in the highest value regardless of its area.
step.type	type of step functions used (default is "left")
xmethod	method for generating the unique set of x-coordinates to examine (see above). Default is "grid" for type="l" or "unique" for type="s".
offset	distance in y-units between the center of the label and the line being labeled. Default is 0.75 times the height of an "m" that would be drawn in a label. For R grid/lattice you must specify offset using the grid unit function, e.g., offset=unit(2,"native") or offset=unit(.25,"cm") ("native" means data units)
xlim	limits for searching for label positions, and is also used to set up plots when pl=TRUE and add=FALSE. Default is total x-axis range for current plot (par("usr")[1:2]). For largest.empty, xlim limits the search for largest rectangles, but it has the same default as above. For pl=TRUE, add=FALSE you may want to extend xlim somewhat to allow large keys to fit, when using keyloc="auto". For drawPlot default is c(0,1). When using largest.empty with ggplot2, xlim and ylim are mandatory.

tilt	set to TRUE to tilt labels to follow the curves, for method="offset" when keys is not given.
window	width of a window, in x-units, to use in determining the local slope for tilting labels. Default is 0.5 times number of characters in the label times the x-width of an "m" in the current character size and font.
npts	number of points to use if xmethod="grid"
cex	character size to pass to text and key. Default is current par("cex"). For putKey, putKeyEmpty, and Points is the size of the plotting symbol.
adj	Default is "auto" which has labcurve figure justification automatically when method="offset". This will cause centering to be used when the local angle of the curve is less than angle.adj.auto in absolute value, left justification if the angle is larger and either the label is under a curve of positive slope or over a curve of negative slope, and right justification otherwise. For step functions, left justification is used when the label is above the curve and right justification otherwise. Set adj=.5 to center labels at computed coordinates. Set to 0 for left-justification, 1 for right. Set adj to a vector to vary adjustments over the curves.
angle.adj.auto	see adj. Does not apply to step functions.
lty	vector of line types which were used to draw the curves. This is only used when keys are drawn. If all of the line types, line widths, and line colors are the same, lines are not drawn in the key.
lwd	vector of line widths which were used to draw the curves. This is only used when keys are drawn. See lty also.
col.	vector of integer color numbers
col	vector of integer color numbers for use in curve labels, symbols, lines, and legends. Default is par("col") for all curves. See lty also.
transparent	Default is TRUE to make key draw transparent legends, i.e., to suppress drawing a solid rectangle background for the legend. Set to FALSE otherwise.
arrow.factor	factor by which to multiply default arrow lengths
point.inc	When keys is a vector of integers, point.inc specifies the x-increment between the point symbols that are overlaid periodically on the curves. By default, point.inc is equal to the range for the x-axis divided by 5.
opts	an optional list which can be used to specify any of the options to labcurve, with the usual element name abbreviations allowed. This is useful when labcurve is being called from another function. Example: opts=list(method="arrow", cex=.8, np=200). For drawPlot a list of labcurve options to pass as labcurve(..., opts=).
key.opts	a list of extra arguments you wish to pass to key(), e.g., key.opts=list(background=1, between=3). The argument names must be spelled out in full.
empty.method	see below
numbins	These two arguments are passed to the largest.empty function's method and numbins arguments (see below). For largest.empty specifies the number of bins in which to discretize both the x and y directions for searching for rectangles. Default is 25.

<code>pl</code>	set to TRUE (or specify <code>add</code> ) to cause the curves in curves to be drawn, under the control of <code>type</code> , <code>lty</code> , <code>lwd</code> , <code>col</code> parameters defined either in the curves lists or in the separate arguments given to <code>labcurve</code> or through <code>opts</code> . For <code>largest.empty</code> , set <code>pl=TRUE</code> to show the rectangle the function found by drawing it with a solid color. May not be used under <code>ggplot2</code> .
<code>add</code>	By default, when curves are actually drawn by <code>labcurve</code> a new plot is started. To add to an existing plot, set <code>add=TRUE</code> .
<code>ylim</code>	When a plot has already been started, <code>ylim</code> defaults to <code>par("usr")[3:4]</code> . When <code>pl=TRUE</code> , <code>ylim</code> and <code>xlim</code> are determined from the ranges of the data. Specify <code>ylim</code> yourself to take control of the plot construction. In some cases it is advisable to make <code>ylim</code> larger than usual to allow for automatically-positioned keys. For <code>largest.empty</code> , <code>ylim</code> specifies the limits on the y-axis to limit the search for rectangle. Here <code>ylim</code> defaults to the same as above, i.e., the range of the y-axis of an open plot from <code>par</code> . For <code>drawPlot</code> the default is <code>c(0, 1)</code> .
<code>xlab</code>	see below
<code>ylab</code>	x-axis and y-axis labels when <code>pl=TRUE</code> and <code>add=FALSE</code> or for <code>drawPlot</code> . Defaults to "" unless the first curve has names for its first two elements, in which case the names of these elements are taken as <code>xlab</code> and <code>ylab</code> .
<code>whichLabel</code>	integer vector corresponding to curves specifying which curves are to be labelled or have a legend
<code>grid</code>	set to TRUE if the R grid package was used to draw the current plot. This prevents <code>labcurve</code> from using <code>par("usr")</code> etc. If using R grid you can pass coordinates and lengths having arbitrary units, as documented in the <code>unit</code> function. This is especially useful for <code>offset</code> .
<code>xrestrict</code>	When having <code>labcurve</code> label curves where they are most separated, you can restrict the search for this separation point to a range of the x-axis, specified as a 2-vector <code>xrestrict</code> . This is useful when one part of the curve is very steep. Even though steep regions may have maximum separation, the labels will collide when curves are steep.
<code>pch</code>	vector of plotting characters for <code>putKey</code> and <code>putKeyEmpty</code> . Can be any value including NA when only a line is used to indentify the group. Is a single plotting character for Points, with the default being the next unused value from among 1, 2, 3, 4, 16, 17, 5, 6, 15, 18, 19.
<code>plot</code>	set to FALSE to keep <code>putKey</code> or <code>putKeyEmpty</code> from actually drawing the key. Instead, the size of the key will be return by <code>putKey</code> , or the coordinates of the key by <code>putKeyEmpty</code> .
<code>ticks</code>	tells <code>drawPlot</code> which axes to draw tick marks and tick labels. Default is "none".
<code>key</code>	for <code>drawPlot</code> and <code>plot.drawPlot</code> . Default is FALSE so that <code>labcurve</code> is used to label points or curves. Set to TRUE to use <code>putKeyEmpty</code> .

## Details

The internal functions `Points`, `Curve`, `Abline` have unique arguments as follows.

**label:** for `Points` and `Curve` is a single character string to label that group of points

**n:** number of points to accept from the mouse. Default is to input points until a right mouse click.



**rug:** for Points. Default is "none" to not show the marginal x or y distributions as rug plots, for the points entered. Other possibilities are used to execute `scat1d` to show the marginal distribution of x, y, or both as rug plots.

**ymean:** for Points, subtracts a constant from each y-coordinate entered to make the overall mean ymean

**degree:** degree of polynomial to fit to points by Curve

**evaluation:** number of points at which to evaluate Bezier curves, polynomials, and other functions in Curve

**ask:** set `ask=TRUE` to give the user the opportunity to try again at specifying points for Bezier curves, step functions, and polynomials

The `labcurve` function used some code from the function `plot.multicurve` written by Rod Tjoelker of The Boeing Company (<tjoelker@espresso.rt.cs.boeing.com>).

If there is only one curve, a label is placed at the middle x-value, and no fancy features such as angle or positive/negative offsets are used.

`key` is called once (with the argument `plot=FALSE`) to find the key dimensions. Then an empty rectangle with at least these dimensions is searched for using `largest.empty`. Then `key` is called again to draw the key there, using the argument `corner=c(.5, .5)` so that the center of the rectangle can be specified to `key`.

If you want to plot the data, an easier way to use `labcurve` is through `xYplot` as shown in some of its examples.

## Value

`labcurve` returns an invisible list with components `x`, `y`, `offset`, `adj`, `cex`, `col`, and if `tilt=TRUE`, `angle`. `offset` is the amount to add to `y` to draw a label. `offset` is negative if the label is drawn below the line. `adj` is a vector containing the values 0, .5, 1.

`largest.empty` returns a list with elements `x` and `y` specifying the coordinates of the center of the rectangle which was found, and element `rect` containing the 4 x and y coordinates of the corners of the found empty rectangle. The area of the rectangle is also returned.

## Author(s)

Frank Harrell  
Department of Biostatistics  
Vanderbilt University  
<fh@fharrell.com>

## See Also

[approx](#), [text](#), [legend](#), [scat1d](#), [xYplot](#), [abline](#)

## Examples

```
n <- 2:8
m <- length(n)
type <- c('1','1','1','1','s','1','1')
```

```

# s=step function l=ordinary line (polygon)
curves <- vector('list', m)

plot(0,1,xlim=c(0,1),ylim=c(-2.5,4),type='n')

set.seed(39)

for(i in 1:m) {
  x <- sort(runif(n[i]))
  y <- rnorm(n[i])
  lines(x, y, lty=i, type=type[i], col=i)
  curves[[i]] <- list(x=x,y=y)
}

labels <- paste('Label for',letters[1:m])
labcurve(curves, labels, tilt=TRUE, type=type, col=1:m)

# Put only single letters on curves at points of
# maximum space, and use key() to define the letters,
# with automatic positioning of the key in the most empty
# part of the plot
# Have labcurve do the plotting, leaving extra space for key

names(curves) <- labels
labcurve(curves, keys=letters[1:m], type=type, col=1:m,
         pl=TRUE, ylim=c(-2.5,4))

# Put plotting symbols at equally-spaced points,
# with a key for the symbols, ignoring line types

labcurve(curves, keys=1:m, lty=1, type=type, col=1:m,
         pl=TRUE, ylim=c(-2.5,4))

# Plot and label two curves, with line parameters specified with data
set.seed(191)
ages.f <- sort(rnorm(50,20,7))
ages.m <- sort(rnorm(40,19,7))
height.f <- pmin(ages.f,21)*.2+60
height.m <- pmin(ages.m,21)*.16+63

labcurve(list(Female=list(ages.f,height.f,col=2),

```

```

      Male =list(ages.m,height.m,col=3,lty='dashed')),
      xlab='Age', ylab='Height', pl=TRUE)
# add ,keys=c('f','m') to label curves with single letters
# For S-Plus use lty=2

# Plot power for testing two proportions vs. n for various odds ratios,
# using 0.1 as the probability of the event in the control group.
# A separate curve is plotted for each odds ratio, and the curves are
# labeled at points of maximum separation

n <- seq(10, 1000, by=10)
OR <- seq(.2,.9,by=.1)
pow <- lapply(OR, function(or,n)list(x=n,y=bpower(p1=.1,odds.ratio=or,n=n)),
             n=n)
names(pow) <- format(OR)
labcurve(pow, pl=TRUE, xlab='n', ylab='Power')

# Plot some random data and find the largest empty rectangle
# that is at least .1 wide and .1 tall

x <- runif(50)
y <- runif(50)
plot(x, y)
z <- largest.empty(x, y, .1, .1)
z
points(z,pch=3) # mark center of rectangle, or
polygon(z$rect, col='blue') # to draw the rectangle, or
#key(z$x, z$y, \dots stuff for legend)

# Use the mouse to draw a series of points using one symbol, and
# two smooth curves or straight lines (if two points are clicked),
# none of these being labeled

# d <- drawPlot(Points(), Curve(), Curve())
# plot(d)

## Not run:
# Use the mouse to draw a Gaussian density, two series of points
# using 2 symbols, one Bezier curve, a step function, and raw data
# along the x-axis as a 1-d scatter plot (rug plot). Draw a key.
# The density function is fit to 3 mouse clicks
# Abline draws a dotted horizontal reference line
d <- drawPlot(Curve('Normal',type='gauss'),
             Points('female'), Points('male'),

```

```

Curve('smooth',ask=TRUE,lty=2), Curve('step',type='s',lty=3),
Points(type='r'), Abline(h=.5, lty=2),
xlab='X', ylab='y', xlim=c(0,100), key=TRUE)
plot(d, ylab='Y')
plot(d, key=FALSE) # label groups using labcurve

## End(Not run)

```

---

label

*Label Attribute of an Object*


---

### Description

`label(x)` retrieves the label attribute of `x`. `label(x) <- "a label"` stores the label attribute, and also puts the class labelled as the first class of `x` (for S-Plus this class is not used and methods for handling this class are not defined so the "label" and "units" attributes are lost upon subsetting). The reason for having this class is so that the subscripting method for labelled, `[labelled]`, can preserve the label attribute in `S`. Also, the print method for labelled objects prefaces the print with the object's label (and units if there). If the variable is also given a "units" attribute using the `units` function, subsetting the variable (using `labelled`) will also retain the "units" attribute.

`label` can optionally append a "units" attribute to the string, and it can optionally return a string or expression (for R's plotmath facility) suitable for plotting. `labelPlotmath` is a function that also has this function, when the input arguments are the 'label' and 'units' rather than a vector having those attributes. When plotmath mode is used to construct labels, the 'label' or 'units' may contain math expressions but they are typed verbatim if they contain percent signs, blanks, or underscores. `labelPlotmath` can optionally create the expression as a character string, which is useful in building `ggplot` commands.

For `Surv` objects, `label` first looks to see if there is an overall "label" attribute for the object, then it looks for saved attributes that `Surv` put in the "inputAttributes" object, looking first at the event variable, then `time2`, and finally `time`. You can restrict the looking by specifying `type`.

`labelLatex` constructs suitable LaTeX labels a variable or from the label and units arguments, optionally right-justifying units if `hfill=TRUE`. This is useful when making tables when the variable in question is not a column heading. If `x` is specified, label and units values are extracted from its attributes instead of from the other arguments.

`Label` (actually `Label.data.frame`) is a function which generates S source code that makes the labels in all the variables in a data frame easy to edit.

`l1ist` is like `list` except that it preserves the names or labels of the component variables in the variables label attribute. This can be useful when looping over variables or using `sapply` or `lapply`. By using `l1ist` instead of `list` one can annotate the output with the current variable's name or label. `l1ist` also defines a `names` attribute for the list and pulls the names from the arguments' expressions for non-named arguments.

`prList` prints a list with element names (without the dollar sign as in default list printing) and if an element of the list is an unclassed list with a name, all of those elements are printed, with titles of the form "primary list name : inner list name". This is especially useful for Rmarkdown html notebooks

when a user-written function creates multiple html and graphical outputs to all be printed in a code chunk. Optionally the names can be printed after the object, and the `htmlfig` option provides more capabilities when making html reports. `prList` does not work for regular html documents.

`putHfig` is similar to `prList` but for a single graphical object that is rendered with a `print` method, making it easy to specify long captions, and short captions for the table of contents in HTML documents. Table of contents entries are generated with the short caption, which is taken as the long caption if there is none. One can optionally not make a table of contents entry. If argument `table=TRUE` table captions will be produced instead. Using `expcoll`, `markupSpecs` html function `expcoll` will be used to make tables expand upon clicking an arrow rather than always appear.

`putHcap` is like `putHfig` except that it assumes that users render the graphics or table outside of the `putHcap` call. This allows things to work in ordinary html documents. `putHcap` does not handle collapsed text.

`plotmathTranslate` is a simple function that translates certain character strings to character strings that can be used as part of R `plotmath` expressions. If the input string has a space or percent inside, the string is surrounded by a call to `plotmath`'s `paste` function.

`as.data.frame.labelled` is a utility function that is called by `[.data.frame]`. It is just a copy of `as.data.frame.vector`. `data.frame.labelled` is another utility function, that adds a class "labelled" to every variable in a data frame that has a "label" attribute but not a "labelled" class.

`relevel.labelled` is a method for preserving labels with the `relevel` function.

`reLabelled` is used to add a 'labelled' class back to variables in data frame that have a 'label' attribute but no 'labelled' class. Useful for changing `cleanup.import()`'d S-Plus data frames back to general form for R and old versions of S-Plus.

## Usage

```
label(x, default=NULL, ...)

## Default S3 method:
label(x, default=NULL, units=plot, plot=FALSE,
      grid=FALSE, html=FALSE, ...)

## S3 method for class 'Surv'
label(x, default=NULL, units=plot, plot=FALSE,
      grid=FALSE, html=FALSE, type=c('any', 'time', 'event'), ...)

## S3 method for class 'data.frame'
label(x, default=NULL, self=FALSE, ...)

label(x, ...) <- value

## Default S3 replacement method:
label(x, ...) <- value

## S3 replacement method for class 'data.frame'
label(x, self=TRUE, ...) <- value
```

```

labelPlotmath(label, units=NULL, plotmath=TRUE, html=FALSE, grid=FALSE,
              chexpr=FALSE)

labelLatex(x=NULL, label='', units='', size='smaller[2]',
           hfill=FALSE, bold=FALSE, default='', double=FALSE)

## S3 method for class 'labelled'
print(x, ...) ## or x - calls print.labelled

Label(object, ...)

## S3 method for class 'data.frame'
Label(object, file='', append=FALSE, ...)

lList(..., labels=TRUE)

prList(x, lcap=NULL, htmlfig=0, after=FALSE)

putHfig(x, ..., scap=NULL, extra=NULL, subsub=TRUE, hr=TRUE,
        table=FALSE, file='', append=FALSE, expcoll=NULL)

putHcap(..., scap=NULL, extra=NULL, subsub=TRUE, hr=TRUE,
        table=FALSE, file='', append=FALSE)

plotmathTranslate(x)

data.frame.labelled(object)

## S3 method for class 'labelled'
relevel(x, ...)

reLabelled(object)

combineLabels(...)

```

### Arguments

<code>x</code>	any object (for <code>plotmathTranslate</code> is a character string). For <code>relevel</code> is a factor variable. For <code>prList</code> is a named list. For <code>putHfig</code> is a graphical object for which a print method will render the graphic (e.g., a <code>ggplot2</code> or <code>plotly</code> object).
<code>self</code>	logical, where to interact with the object or its components
<code>units</code>	set to <code>TRUE</code> to append the 'units' attribute (if present) to the returned label. The 'units' are surrounded by brackets. For <code>labelPlotmath</code> and <code>labelLatex</code> is a character string containing the units of measurement. When <code>plot</code> is <code>TRUE</code> , <code>units</code> defaults to <code>TRUE</code> .
<code>plot</code>	set to <code>TRUE</code> to return a label suitable for R's <code>plotmath</code> facility (returns an expression instead of a character string) if R is in effect. If <code>units</code> is also <code>TRUE</code> , and if

	both 'label' and 'units' attributes are present, the 'units' will appear after the label but in smaller type and will not be surrounded by brackets.
default	if x does not have a 'label' attribute and default (a character string) is specified, the label will be taken as default. For <code>labelLatex</code> the default is the name of the first argument if it is a variable and not a label.
grid	Currently R's <code>lattice</code> and <code>grid</code> functions do not support <code>plotmath</code> expressions for <code>xlab</code> and <code>ylab</code> arguments. When using <code>lattice</code> functions in R, set the argument <code>grid</code> to TRUE so that <code>labelPlotmath</code> can return an ordinary character string instead of an expression.
html	set to TRUE to use HTML formatting instead of <code>plotmath</code> expressions for constructing labels with units
type	for <code>Surv</code> objects specifies the type of element for which to restrict the search for a label
label	a character string containing a variable's label
plotmath	set to TRUE to have <code>labelMathplot</code> return an expression for plotting using R's <code>plotmath</code> facility. If R is not in effect, an ordinary character string is returned.
chexpr	set to TRUE to have <code>labelPlotmath</code> return a character string of the form "expression(...)"
size	LaTeX size for units. Default is two sizes smaller than <code>label</code> , which assumes that the LaTeX <code>resize</code> package is in use.
hfill	set to TRUE to right-justify units in the field. This is useful when multiple labels are being put into rows in a LaTeX <code>tabular</code> environment, and will cause a problem if the label is used in an environment where <code>hfill</code> is not appropriate.
bold	set to TRUE to have <code>labelLatex</code> put the label in bold face.
double	set to TRUE to represent backslash in LaTeX as four backslashes in place of two. This is needed if, for example, you need to convert the result using <code>as.formula</code>
value	the label of the object, or "".
object	a data frame
...	a list of variables or expressions to be formed into a list. Ignored for <code>print.labelled</code> . For <code>relevel</code> is the level (a single character string) to become the new reference (first) category. For <code>putHfig</code> and <code>putHcap</code> represents one or more character strings that are pasted together, separated by a blank.
file	the name of a file to which to write S source code. Default is "", meaning standard output. For <code>putHcap</code> , set <code>file</code> to FALSE to return a character vector instead of writing to file.
append	set to TRUE to append code generated by <code>Label</code> to file <code>file</code> . Also used for <code>putHfig</code> , <code>putHcap</code> .
labels	set to FALSE to make <code>l1ist</code> ignore the variables' <code>label</code> attribute and use the variables' names.
lcap	an optional vector of character strings corresponding to elements in <code>x</code> for <code>prList</code> . These contain long captions that do not appear in the table of contents but which are printed right after the short caption in the body, in the same font.
htmlfig	for <code>prList</code> set to 1 to use HTML markup by running the object names through <code>markupSpecs\$html\$cap</code> for figure captions. Set <code>htmlfig=2</code> to also preface the figure caption with "### " so that it will appear in the table of contents.

after	set to TRUE to have prList put names after the printed object instead of before
scap	a character string specifying the short (or possibly only) caption.
extra	an optional vector of character strings. When present the long caption will be put in the first column of an HTML table and the elements of extra in subsequent columns. This allows extra information to appear in the long caption in a way that is right-justified to the right of the flowing caption text.
subsub	set to FALSE to suppress "### " from being placed in front of the short caption. Set it to different character string to use that instead. Set it to "" to ignore short captions entirely. For example to use second-level headings for the table of contents specify subsub="## ".
hr	applies if a caption is present. Specify FALSE to not put a horizontal line before the caption and figure.
table	set to TRUE to produce table captions instead of figure captions
expcoll	character string to be visible, with a clickable arrow following to allow initial hiding of a table and its captions. Cannot be used with table=FALSE.

### Value

label returns the label attribute of x, if any; otherwise, "". label is used most often for the individual variables in data frames. The function sas.get copies labels over from SAS if they exist.

### See Also

[sas.get](#), [describe](#), [extractlabs](#), [hlab](#)

### Examples

```
age <- c(21,65,43)
y <- 1:3
label(age) <- "Age in Years"
plot(age, y, xlab=label(age))

data <- data.frame(age=age, y=y)
label(data)

label(data, self=TRUE) <- "A data frame"
label(data, self=TRUE)

x1 <- 1:10
x2 <- 10:1
label(x2) <- 'Label for x2'
units(x2) <- 'mmHg'
x2
x2[1:5]
dframe <- data.frame(x1, x2)
Label(dframe)

labelLatex(x2, hfill=TRUE, bold=TRUE)
```



```
labelLatex(label='Velocity', units='m/s')

##In these examples of llist, note that labels are printed after
##variable names, because of print.labelled
a <- 1:3
b <- 4:6
label(b) <- 'B Label'
llist(a,b)
llist(a,b,d=0)
llist(a,b,0)

w <- llist(a, b>5, d=101:103)
sapply(w, function(x){
  hist(as.numeric(x), xlab=label(x))
  # locator(1) ## wait for mouse click
})

# Or: for(u in w) {hist(u); title(label(u))}
```

---

Lag

*Lag a Numeric, Character, or Factor Vector*

---

### Description

Shifts a vector shift elements later. Character or factor variables are padded with "", numerics with NA. The shift may be negative.

### Usage

```
Lag(x, shift = 1)
```

### Arguments

x	a vector
shift	integer specifying the number of observations to be shifted to the right. Negative values imply shifts to the left.

### Details

Attributes of the original object are carried along to the new lagged one.

### Value

a vector like x

### Author(s)

Frank Harrell

**See Also**[lag](#)**Examples**

```
Lag(1:5,2)
Lag(letters[1:4],2)
Lag(factor(letters[1:4]),-2)
# Find which observations are the first for a given subject
id <- c('a','a','b','b','b','c')
id != Lag(id)
!duplicated(id)
```

---

`latestFile`*latestFile*

---

**Description**

Find File With Latest Modification Time

**Usage**

```
latestFile(pattern, path = ".", verbose = TRUE)
```

**Arguments**

<code>pattern</code>	a regular expression; see <a href="#">base::list.files()</a>
<code>path</code>	full path, defaulting to current working directory
<code>verbose</code>	set to FALSE to not report on total number of matching files

**Details**

Subject to matching on `pattern` finds the last modified file, and if `verbose` is TRUE reports on how many total files matched `pattern`.

**Value**

the name of the last modified file

**Author(s)**

Frank Harrell

**See Also**[base::list.files\(\)](#)

**Description**

`latex` converts its argument to a `.tex` file appropriate for inclusion in a LaTeX2e document. `latex` is a generic function that calls one of `latex.default`, `latex.function`, `latex.list`.

`latex.default` does appropriate rounding and decimal alignment and produces a file containing a LaTeX tabular environment to print the matrix or data.frame `x` as a table.

`latex.function` prepares an S function for printing by issuing sed commands that are similar to those in the `S.to.latex` procedure in the `s.to.latex` package (Chambers and Hastie, 1993). `latex.function` can also produce verbatim output or output that works with the Sweave LaTeX style.

`latex.list` calls `latex` recursively for each element in the argument.

`latexTranslate` translates particular items in character strings to LaTeX format, e.g., makes `'a^2 = a\^2\$'` for superscript within variable labels. LaTeX names of greek letters (e.g., "alpha") will have backslashes added if `greek==TRUE`. Math mode is inserted as needed. `latexTranslate` assumes that input text always has matches, e.g. `[ ] ( )`, and that surrounding by `'\$\$'` is OK.

`htmlTranslate` is similar to `latexTranslate` but for html translation. It doesn't need math mode and assumes dollar signs are just that.

`latexSN` converts a vector floating point numbers to character strings using LaTeX exponents. Dollar signs to enter math mode are not added. Similarly, `htmlSN` converts to scientific notation in html.

`latexVerbatim` on an object executes the object's `print` method, capturing the output for a file inside a LaTeX verbatim environment.

`dvi` uses the system `latex` command to compile LaTeX code produced by `latex`, including any needed styles. `dvi` will put a `'\documentclass{report}'` and `'\end{document}'` wrapper around a file produced by `latex`. By default, the 'geometry' LaTeX package is used to omit all margins and to set the paper size to a default of 5.5in wide by 7in tall. The result of `dvi` is a `.dvi` file. To both format and screen display a non-default size, use for example `print(dvi(latex(x), width=3, height=4), width=3, height=4)`. Note that you can use something like `'xdvi -geometry 460x650 -margins 2.25in file'` without changing LaTeX defaults to emulate this.

`dvips` will use the system `dvips` command to print the `.dvi` file to the default system printer, or create a postscript file if `file` is specified.

`dvigv` uses the system `dvips` command to convert the input object to a `.dvi` file, and uses the system `dvips` command to convert it to postscript. Then the postscript file is displayed using Ghostview (assumed to be the system command `gv`).

There are show methods for displaying typeset LaTeX on the screen using the system `xdvi` command. If you show a LaTeX file created by `latex` without running it through `dvi` using `show.dvi(object)`, the show method will run it through `dvi` automatically. These show methods are not S Version 4 methods so you have to use full names such as `show.dvi` and `show.latex`. Use the `print` methods for more automatic display of typesetting, e.g. typing `latex(x)` will invoke `xdvi` to view the typeset document.

**Usage**

```

latex(object, ...)

## Default S3 method:
latex(object,
      title=first.word(deparse(substitute(object))),
      file=paste(title, ".tex", sep=""),
      append=FALSE, label=title,
      rowlabel=title, rowlabel.just="l",
      cgroup=NULL, n.cgroup=NULL,
      rgroup=NULL, n.rgroup=NULL,
      cgroupTexCmd="bfseries",
      rgroupTexCmd="bfseries",
      rownamesTexCmd=NULL,
      colnamesTexCmd=NULL,
      cellTexCmds=NULL,
      rowname, cgroup.just=rep("c",length(n.cgroup)),
      colheads=NULL,
      extracolheads=NULL, extracolsize='scriptsize',
      dcolumn=FALSE, numeric.dollar=!dcolumn, cdot=FALSE,
      longtable=FALSE, draft.longtable=TRUE, ctable=FALSE, booktabs=FALSE,
      table.env=TRUE, here=FALSE, lines.page=40,
      caption=NULL, caption.lot=NULL, caption.loc=c('top','bottom'),
      star=FALSE,
      double.slash=FALSE,
      vbar=FALSE, collabel.just=rep("c",nc), na.blank=TRUE,
      insert.bottom=NULL, insert.bottom.width=NULL,
      insert.top=NULL,
      first.hline.double=!(booktabs | ctable),
      where='!tbp', size=NULL,
      center=c('center','centering','centerline','none'),
      landscape=FALSE,
      multicol=TRUE,
      math.row.names=FALSE, already.math.row.names=FALSE,
      math.col.names=FALSE, already.math.col.names=FALSE,
      hyperref=NULL, continued='continued',
      ...) # x is a matrix or data.frame

## S3 method for class 'function'
latex(
  object,
  title=first.word(deparse(substitute(object))),
  file=paste(title, ".tex", sep=""),
  append=FALSE,
  assignment=TRUE, type=c('example','verbatim','Sinput'),
  width.cutoff=70, size='', ...)

## S3 method for class 'list'

```

```

latex(
  object,
  title=first.word(deparse(substitute(object))),
  file=paste(title, ".tex", sep=""),
  append=FALSE,
  label,
  caption,
  caption.lot,
  caption.loc=c('top','bottom'),
  ...)

## S3 method for class 'latex'
print(x, ...)

latexTranslate(object, inn=NULL, out=NULL, pb=FALSE, greek=FALSE, na='',
  ...)

htmlTranslate(object, inn=NULL, out=NULL, greek=FALSE, na='',
  code=htmlSpecialType(), ...)

latexSN(x)

htmlSN(x, pretty=TRUE, ...)

latexVerbatim(x, title=first.word(deparse(substitute(x))),
  file=paste(title, ".tex", sep=""),
  append=FALSE, size=NULL, hspace=NULL,
  width=.Options$width, length=.Options$length, ...)

dvi(object, ...)
## S3 method for class 'latex'
dvi(object, prlog=FALSE, nomargins=TRUE, width=5.5, height=7, ...)
## S3 method for class 'dvi'
print(x, ...)
dvips(object, ...)
## S3 method for class 'latex'
dvips(object, ...)
## S3 method for class 'dvi'
dvips(object, file, ...)
## S3 method for class 'latex'
show(object) # or show.dvi(object) or just object
dviGV(object, ...)
## S3 method for class 'latex'
dviGV(object, ...) # or gvDvi(dvi(object))
## S3 method for class 'dvi'
dviGV(object, ...)

```

**Arguments**

object	For latex, any S object. For dvi or dvigv, an object created by latex. For latexTranslate is a vector of character strings to translate. Any NAs are set to blank strings before conversion.
x	any object to be printed verbatim for latexVerbatim. For latexSN or htmlSN, x is a numeric vector.
title	name of file to create without the '.tex' extension. If this option is not set, value/string of x (see above) is printed in the top left corner of the table. Set title='' to suppress this output.
file	name of the file to create. The default file name is 'x.tex' where x is the first word in the name of the argument for x. Set file="" to have the generated LaTeX code just printed to standard output. This is especially useful when running under Sweave in R using its 'results=tex' tag, to save having to manage many small external files. When file="", latex keeps track of LaTeX styles that are called for by creating or modifying an object latexStyles (in .GlobalTemp in R or in frame 0 in S-Plus). latexStyles is a vector containing the base names of all the unique LaTeX styles called for so far in the current session. See the end of the examples section for a way to use this object to good effect. For dvips, file is the name of an output postscript file.
append	defaults to FALSE. Set to TRUE to append output to an existing file.
label	a text string representing a symbolic label for the table for referencing in the LaTeX '\label' and '\ref' commands. label is only used if caption is given.
rowlabel	If x has row dimnames, rowlabel is a character string containing the column heading for the row dimnames. The default is the name of the argument for x.
rowlabel.just	If x has row dimnames, specifies the justification for printing them. Possible values are "l", "r", "c". The heading (rowlabel) itself is left justified if rowlabel.just="l", otherwise it is centered.
cgroup	a vector of character strings defining major column headings. The default is to have none.
n.cgroup	a vector containing the number of columns for which each element in cgroup is a heading. For example, specify cgroup=c("Major 1", "Major 2"), n.cgroup=c(3,3) if "Major 1" is to span columns 1-3 and "Major 2" is to span columns 4-6. rowlabel does not count in the column numbers. You can omit n.cgroup if all groups have the same number of columns.
rgroup	a vector of character strings containing headings for row groups. n. rgroup must be present when rgroup is given. The first n. rgroup[1] rows are sectioned off and rgroup[1] is used as a bold heading for them. The usual row dimnames (which must be present if rgroup is) are indented. The next n. rgroup[2] rows are treated likewise, etc.
n. rgroup	integer vector giving the number of rows in each grouping. If rgroup is not specified, n. rgroup is just used to divide off blocks of rows by horizontal lines. If rgroup is given but n. rgroup is omitted, n. rgroup will default so that each row group contains the same number of rows.

cgroupTexCmd	A character string specifying a LaTeX command to be used to format column group labels. The default, "bfseries", sets the current font to 'bold'. It is possible to supply a vector of strings so that each column group label is formatted differently. Please note that the first item of the vector is used to format the title (even if a title is not used). Currently the user needs to handle these issue. Multiple effects can be achieved by creating custom LaTeX commands; for example, "\providecommand{\redscshape}{\color{red}\scshape}" creates a LaTeX command called '\redscshape' that formats the text in red small-caps.
rgroupTexCmd	A character string specifying a LaTeX command to be used to format row group labels. The default, "bfseries", sets the current font to 'bold'. A vector of strings can be supplied to format each row group label differently. Normal recycling applies if the vector is shorter than n.rgroups. See also cgroupTexCmd above regarding multiple effects.
rownamesTexCmd	A character string specifying a LaTeX command to be used to format rownames. The default, NULL, applies no command. A vector of different commands can also be supplied. See also cgroupTexCmd above regarding multiple effects.
colnamesTexCmd	A character string specifying a LaTeX command to be used to format column labels. The default, NULL, applies no command. It is possible to supply a vector of strings to format each column label differently. If column groups are not used, the first item in the vector will be used to format the title. Please note that if column groups are used the first item of cgroupTexCmd and not colnamesTexCmd is used to format the title. The user needs to allow for these issues when supplying a vector of commands. See also cgroupTexCmd above regarding multiple effects.
cellTexCmds	A matrix of character strings which are LaTeX commands to be used to format each element, or cell, of the object. The matrix must have the same NROW() and NCOL() as the object. The default, NULL, applies no formats. Empty strings also apply no formats, and one way to start might be to create a matrix of empty strings with <code>matrix(rep("", NROW(x) * NCOL(x)), nrow=NROW(x))</code> and then selectively change appropriate elements of the matrix. Note that you might need to set <code>numeric.dollar=FALSE</code> (to disable math mode) for some effects to work. See also cgroupTexCmd above regarding multiple effects.
na.blank	Set to TRUE to use blanks rather than NA for missing values. This usually looks better in latex.
insert.bottom	an optional character string to typeset at the bottom of the table. For "ctable" style tables, this is placed in an unmarked footnote.
insert.bottom.width	character string; a tex width controlling the width of the insert.bottom text. Currently only does something with using <code>longtable=TRUE</code> .
insert.top	a character string to insert as a heading right before beginning tabular environment. Useful for multiple sub-tables.
first.hline.double	set to FALSE to use single horizontal rules for styles other than "bookmark" or "ctable"
rowname	rownames for tabular environment. Default is rownames of matrix or data.frame. Specify rowname=NULL to suppress the use of row names.

<code>cgroup.just</code>	justification for labels for column groups. Defaults to "c".
<code>colheads</code>	a character vector of column headings if you don't want to use <code>dimnames(object)[[2]]</code> . Specify <code>colheads=FALSE</code> to suppress column headings.
<code>extracolheads</code>	an optional vector of extra column headings that will appear under the main headings (e.g., sample sizes). This character vector does not need to include an empty space for any rowname in effect, as this will be added automatically. You can also form subheadings by splitting character strings defining the column headings using the usual backslash n newline character.
<code>extracolsize</code>	size for <code>extracolheads</code> or for any second lines in column names; default is "scriptsize"
<code>dcolumn</code>	see <a href="#">format.df</a>
<code>numeric.dollar</code>	logical, default <code>!dcolumn</code> . Set to TRUE to place dollar signs around numeric values when <code>dcolumn=FALSE</code> . This assures that latex will use minus signs rather than hyphens to indicate negative numbers. Set to FALSE when <code>dcolumn=TRUE</code> , as <code>dcolumn.sty</code> automatically uses minus signs.
<code>math.row.names</code>	logical, set true to place dollar signs around the row names.
<code>already.math.row.names</code>	set to TRUE to prevent any math mode changes to row names
<code>math.col.names</code>	logical, set true to place dollar signs around the column names.
<code>already.math.col.names</code>	set to TRUE to prevent any math mode changes to column names
<code>hyperref</code>	if <code>table.env=TRUE</code> is a character string used to generate a LaTeX hyperref enclosure
<code>continued</code>	a character string used to indicate pages after the first when making a long table
<code>cdot</code>	see <a href="#">format.df</a>
<code>longtable</code>	Set to TRUE to use David Carlisle's LaTeX <code>longtable</code> style, allowing long tables to be split over multiple pages with headers repeated on each page. The "style" element is set to "longtable". The latex '\usepackage' must reference '[longtable]'. The file 'longtable.sty' will need to be in a directory in your TEXINPUTS path.
<code>draft.longtable</code>	I forgot what this does.
<code>ctable</code>	set to TRUE to use Wybo Dekker's 'ctable' style from CTAN. Even though for historical reasons it is not the default, it is generally the preferred method. Thicker but not doubled '\hline's are used to start a table when <code>ctable</code> is in effect.
<code>booktabs</code>	set <code>booktabs=TRUE</code> to use the 'booktabs' style of horizontal rules for better tables. In this case, double '\hline's are not used to start a table.
<code>table.env</code>	Set <code>table.env=FALSE</code> to suppress enclosing the table in a LaTeX 'table' environment. <code>table.env</code> only applies when <code>longtable=FALSE</code> . You may not specify a caption if <code>table.env=FALSE</code> .
<code>here</code>	Set to TRUE if you are using <code>table.env=TRUE</code> with <code>longtable=FALSE</code> and you have installed David Carlisle's 'here.sty' LaTeX style. This will cause the LaTeX 'table' environment to be set up with option 'H' to guarantee that the table



will appear exactly where you think it will in the text. The "style" element is set to "here". The latex '\usepackage' must reference '[here]'. The file 'here.sty' will need to be in a directory in your TEXINPUTS path. 'here' is largely obsolete with LaTeX2e.

lines.page	Applies if longtable=TRUE. No more than lines.page lines in the body of a table will be placed on a single page. Page breaks will only occur at rgroup boundaries.
caption	a text string to use as a caption to print at the top of the first page of the table. Default is no caption.
caption.lot	a text string representing a short caption to be used in the "List of Tables". By default, LaTeX will use caption. If you get inexplicable 'latex' errors, you may need to supply caption.lot to make the errors go away.
caption.loc	set to "bottom" to position a caption below the table instead of the default of "top".
star	apply the star option for ctables to allow a table to spread over two columns when in twocolumn mode.
double.slash	set to TRUE to output ""\" as "\\\" in LaTeX commands. Useful when you are reading the output file back into an S vector for later output.
vbar	logical. When vbar==TRUE, columns in the tabular environment are separated with vertical bar characters. When vbar==FALSE, columns are separated with white space. The default, vbar==FALSE, produces tables consistent with the style sheet for the Journal of the American Statistical Association.
collabel.just	justification for column labels.
assignment	logical. When TRUE, the default, the name of the function and the assignment arrow are printed to the file.
where	specifies placement of floats if a table environment is used. Default is "!tbp". To allow tables to appear in the middle of a page of text you might specify where="!htbp" to latex.default.
size	size of table text if a size change is needed (default is no change). For example you might specify size="small" to use LaTeX font size "small". For latex.function is a character string that will be appended to "Sinput" such as "small".
center	default is "center" to enclose the table in a 'center' environment. Use center="centering" or "centerline" to instead use LaTeX 'centering' or centerline directives, or center="none" to use no centering. centerline can be useful when objects besides a tabular are enclosed in a single table environment. This option was implemented by Markus Jäntti <markus.jantti@iki.fi> of Abo Akademi University.
landscape	set to TRUE to enclose the table in a 'landscape' environment. When ctable is TRUE, will use the rotate argument to ctable.
type	The default uses the S alltt environment for latex.function, Set type="verbatim" to instead use the LaTeX 'verbatim' environment. Use type="Sinput" if using Sweave, especially if you have customized the Sinput environment, for example using the Sweave style which uses the listings LaTeX package.

<code>width.cutoff</code>	width of function text output in columns; see <code>deparse</code>
<code>...</code>	other arguments are accepted and ignored except that <code>latex</code> passes arguments to <code>format.df</code> (e.g., <code>col.just</code> and other formatting options like <code>dec</code> , <code>rdec</code> , and <code>cdec</code> ). For <code>latexVerbatim</code> these arguments are passed to the <code>print</code> function. Ignored for <code>latexTranslate</code> and <code>htmlTranslate</code> . For <code>htmlSN</code> , these arguments are passed to <code>prettyNum</code> or <code>format</code> .
<code>inn, out</code>	specify additional input and translated strings over the usual defaults
<code>pb</code>	If <code>pb=TRUE</code> , <code>latexTranslate</code> also translates <code>'[()]</code> to math mode using <code>'\left, \right'</code> .
<code>greek</code>	set to <code>TRUE</code> to have <code>latexTranslate</code> put names for greek letters in math mode and add backslashes. For <code>htmlTranslate</code> , translates greek letters to corresponding html characters, ignoring "modes".
<code>na</code>	single character string to translate NA values to for <code>latexTranslate</code> and <code>htmlTranslate</code>
<code>code</code>	set to <code>'unicode'</code> to use HTML unicode characters or <code>'&amp;'</code> to use the ampersand pound number format
<code>pretty</code>	set to <code>FALSE</code> to have <code>htmlSN</code> use <code>format</code> instead of <code>prettyNum</code>
<code>hspace</code>	horizontal space, e.g., extra left margin for verbatim text. Default is none. Use e.g. <code>hspace="10ex"</code> to add 10 extra spaces to the left of the text.
<code>length</code>	for S-Plus only; is the length of the output page for printing and capturing verbatim text
<code>width, height</code>	are the <code>options()</code> to have in effect only for when <code>print</code> is executed. Defaults are current options. For <code>dvi</code> these specify the paper width and height in inches if <code>nomargins=TRUE</code> , with defaults of 5.5 and 7, respectively.
<code>prlog</code>	set to <code>TRUE</code> to have <code>dvi</code> print, to the S-Plus session, the LaTeX <code>.log</code> file.
<code>multicol</code>	set to <code>FALSE</code> to not use <code>'\multicolumn'</code> in header of table
<code>nomargins</code>	set to <code>FALSE</code> to use default LaTeX margins when making the <code>.dvi</code> file

## Details

`latex.default` optionally outputs a LaTeX comment containing the calling statement. To output this comment, run `options(omitlatexcom=FALSE)` before running. The default behavior or suppressing the comment is helpful when running RMarkdown to produce pdf output using LaTeX, as this uses pandoc which is fooled into try to escape the percent comment symbol.

If running under Windows and using MikTeX, `latex` and `yap` must be in your system path, and `yap` is used to browse `'dvi'` files created by `latex`. You should install the `'geometry.sty'` and `'ctable.sty'` styles in MikTeX to make optimum use of `latex()`.

On Mac OS X, you may have to append the `'/usr/texbin'` directory to the system path. Thanks to Kevin Thorpe (<kevin.thorpe@utoronto.ca>) one way to set up Mac OS X is to install `'X11'` and `'X11SDK'` if not already installed, start `'X11'` within the R GUI, and issue the command `Sys.setenv(PATH=paste(Sys.getenv("PATH"), "/usr/texbin", sep=":"))`. To avoid any complications of using `'X11'` under MacOS, users can install the `'TeXShop'` package, which will associate `'dvi'` files with a viewer that displays a `'pdf'` version of the file after a hidden conversion from `'dvi'` to `'pdf'`.

System options can be used to specify external commands to be used. Defaults are given by `options(xdviCmd='xdvi')` or `options(xdviCmd='yap')`, `options(dvipscmd='dvips')`, `options(latexCmd='latex')`. For MacOS specify `options(xdviCmd='MacdviX')` or if TeXShop is installed, `options(xdviCmd='open')`.

To use 'pdflatex' rather than 'latex', set `options(latexCmd='pdflatex')`, `options(dviExtension='pdf')`, and set `options('xdviCmd')` to your chosen PDF previewer.

If running S-Plus and your directory for temporary files is not '/tmp' (Unix/Linux) or '\\windows\\temp' (Windows), add your own `tempdir` function such as `tempdir <- function() "/yourmaindirectory/yoursubdirectory"`

To prevent the latex file from being displayed store the result of latex in an object, e.g. `w <- latex(object, file='foo.tex')`.

### Value

`latex` and `dvi` return a list of class `latex` or `dvi` containing character string elements `file` and `style`. `file` contains the name of the generated file, and `style` is a vector (possibly empty) of styles to be included using the LaTeX2e '\usepackage' command.

`latexTranslate` returns a vector of character strings

### Side Effects

creates various system files and runs various Linux/UNIX system commands which are assumed to be in the system path.

### Author(s)

Frank E. Harrell, Jr.,  
Department of Biostatistics,  
Vanderbilt University,  
<fh@fharrell.com>

Richard M. Heiberger,  
Department of Statistics,  
Temple University, Philadelphia, PA.  
<rmh@temple.edu>

David R. Whiting,  
School of Clinical Medical Sciences (Diabetes),  
University of Newcastle upon Tyne, UK.  
<david.whiting@ncl.ac.uk>

### See Also

[html](#), [format.df](#), [texi2dvi](#)

### Examples

```
x <- matrix(1:6, nrow=2, dimnames=list(c('a','b'),c('c','d','this that')))  
## Not run:  
latex(x) # creates x.tex in working directory  
# The result of the above command is an object of class "latex"  
# which here is automatically printed by the latex print method.
```

```

# The latex print method prepends and appends latex headers and
# calls the latex program in the PATH. If the latex program is
# not in the PATH, you will get error messages from the operating
# system.

w <- latex(x, file='/tmp/my.tex')
# Does not call the latex program as the print method was not invoked
print.default(w)
# Shows the contents of the w variable without attempting to latex it.

d <- dvi(w) # compile LaTeX document, make .dvi
           # latex assumed to be in path
d         # or show(d) : run xdvi (assumed in path) to display
w         # or show(w) : run dvi then xdvi
dvips(d)  # run dvips to print document
dvips(w)  # run dvi then dvips
library(tools)
texi2dvi('/tmp/my.tex') # compile and produce pdf file in working dir.

## End(Not run)
latex(x, file="") # just write out LaTeX code to screen

## Not run:
# Use paragraph formatting to wrap text to 3 in. wide in a column
d <- data.frame(x=1:2,
               y=c(paste("a",
                        paste(rep("very",30),collapse=" "),"long string"),
                        "a short string"))
latex(d, file="", col.just=c("l", "p{3in}"), table.env=FALSE)

## End(Not run)

## Not run:
# After running latex( ) multiple times with different special styles in
# effect, make a file that will call for the needed LaTeX packages when
# latex is run (especially when using Sweave with R)
if(exists(latexStyles))
  cat(paste('\usepackage{',latexStyles,'}',sep=' '),
      file='stylesused.tex', sep='\n')
# Then in the latex job have something like:
# \documentclass{article}
# \input{stylesused}
# \begin{document}
# ...

## End(Not run)

```

**Description**

Check whether the options for latex functions have been specified. If any of `options()["latexcmd", "dviExtension", "xdvcmd"]` are NULL, an error message is displayed.

**Usage**

```
latexCheckOptions(...)
```

**Arguments**

... Any arguments are ignored.

**Value**

If any NULL options are detected, the invisible text of the error message. If all three options have non-NULL values, NULL.

**Author(s)**

Richard M. Heiberger <rmh@temple.edu>

**See Also**

[latex](#)

---

latexDotchart

*Enhanced Dot Chart for LaTeX Picture Environment with epic*

---

**Description**

latexDotchart is a translation of the dotchart3 function for producing a vector of character strings containing LaTeX picture environment markup that mimics dotchart3 output. The LaTeX epic and color packages are required. The add and horizontal=FALSE options are not available for latexDotchart, however.

**Usage**

```
latexDotchart(data, labels, groups=NULL, gdata=NA,
  xlab='', auxdata, auxgdata=NULL, auxtitle,
  w=4, h=4, margin,
  lines=TRUE, dotsize = .075, size='small', size.labels='small',
  size.group.labels='normalsize', ttlables=FALSE, sort.=TRUE,
  xaxis=TRUE, lcolor='gray', ...)
```

**Arguments**

<code>data</code>	a numeric vector whose values are shown on the x-axis
<code>labels</code>	a vector of labels for each point, corresponding to <code>x</code> . If omitted, <code>names(data)</code> are used, and if there are no names, integers prefixed by <code>"#"</code> are used.
<code>groups</code>	an optional categorical variable indicating how data values are grouped
<code>gdata</code>	data values for groups, typically summaries such as group medians
<code>xlab</code>	x-axis title
<code>auxdata</code>	a vector of auxiliary data, of the same length as the first ( <code>data</code> ) argument. If present, this vector of values will be printed outside the right margin of the dot chart. Usually <code>auxdata</code> represents cell sizes.
<code>auxgdata</code>	similar to <code>auxdata</code> but corresponding to the <code>gdata</code> argument. These usually represent overall sample sizes for each group of lines.
<code>auxtitle</code>	if <code>auxdata</code> is given, <code>auxtitle</code> specifies a column heading for the extra printed data in the chart, e.g., <code>"N"</code>
<code>w</code>	width of picture in inches
<code>h</code>	height of picture in inches
<code>margin</code>	a 4-vector representing, in inches, the margin to the left of the x-axis, below the y-axis, to the right of the x-axis, and above the y-axis. By default these are computed making educated cases about how to accommodate <code>auxdata</code> etc.
<code>lines</code>	set to <code>FALSE</code> to suppress drawing of reference lines
<code>dotsize</code>	diameter of filled circles, in inches, for drawing dots
<code>size</code>	size of text in picture. This and the next two arguments are LaTeX font commands without the opening backslash, e.g., <code>'normalsize'</code> , <code>'small'</code> , <code>'large'</code> , <code>smaller[2]</code> .
<code>size.labels</code>	size of labels
<code>size.group.labels</code>	size of labels corresponding to groups
<code>ttlables</code>	set to <code>TRUE</code> to use typewriter monospaced font for labels
<code>sort.</code>	set to <code>FALSE</code> to keep <code>latexDotchart</code> from sorting the input data, i.e., it will assume that the data are already properly arranged. This is especially useful when you are using <code>gdata</code> and <code>groups</code> and you want to control the order that groups appear on the chart (from top to bottom).
<code>xaxis</code>	set to <code>FALSE</code> to suppress drawing x-axis
<code>lcolor</code>	color for horizontal reference lines. Default is <code>"gray"</code>
<code>...</code>	ignored

**Author(s)**

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>

**See Also**[dotchart3](#)**Examples**

```
## Not run:
z <- latexDotchart(c(.1,.2), c('a','bbAAb'), xlab='This Label',
                  auxdata=c(.1,.2), auxtitle='Zcriteria')
f <- '/tmp/t.tex'
cat('\documentclass{article}\n\usepackage{epic,color}\n\begin{document}\n', file=f)
cat(z, sep='\n', file=f, append=TRUE)
cat('\end{document}\n', file=f, append=TRUE)

set.seed(135)
maj <- factor(c(rep('North',13),rep('South',13)))
g <- paste('Category',rep(letters[1:13],2))
n <- sample(1:15000, 26, replace=TRUE)
y1 <- runif(26)
y2 <- pmax(0, y1 - runif(26, 0, .1))
z <- latexDotchart(y1, g, groups=maj, auxdata=n, auxtitle='n', xlab='Y',
                  size.group.labels='large', ttlabs=TRUE)
f <- '/tmp/t2.tex'
cat('\documentclass{article}\n\usepackage{epic,color}\n\begin{document}\n\framebox{' , file=f)
cat(z, sep='\n', file=f, append=TRUE)
cat('}\end{document}\n', file=f, append=TRUE)

## End(Not run)
```

---

 latexTabular

---

*Convert a Data Frame or Matrix to a LaTeX Tabular*


---

**Description**

latexTabular creates a character vector representing a matrix or data frame in a simple ‘tabular’ environment.

**Usage**

```
latexTabular(x, headings=colnames(x),
            align =paste(rep('c',ncol(x)),collapse=' '),
            halign=paste(rep('c',ncol(x)),collapse=' '),
            helvetica=TRUE, translate=TRUE, hline=0, center=FALSE, ...)
```

**Arguments**

**x** a matrix or data frame, or a vector that is automatically converted to a matrix

**headings** a vector of character strings specifying column headings for ‘latexTabular’, defaulting to x’s colnames. To make multi-line headers use the newline character inside elements of headings.

align	a character strings specifying column alignments for 'latexTabular', defaulting to <code>paste(rep('c', ncol(x)), collapse='')</code> to center. You may specify <code>align='c c'</code> and other LaTeX tabular formatting.
halign	a character strings specifying alignment for column headings, defaulting to centered.
helvetica	set to FALSE to use default LaTeX font in 'latexTabular' instead of helvetica.
translate	set to FALSE if column headings and table entries are already in LaTeX format, otherwise latexTabular will run them through latexTranslate
hline	set to 1 to put hline after heading, 2 to also put hlines before and after heading and at table end
center	set to TRUE to enclose the tabular in a LaTeX center environment
...	if present, x is run through <code>format.df</code> with those extra arguments

**Value**

a character string containing LaTeX markup

**Author(s)**

Frank E. Harrell, Jr.,  
 Department of Biostatistics,  
 Vanderbilt University,  
 <fh@fharrell.com>

**See Also**

[latex.default](#), [format.df](#)

**Examples**

```
x <- matrix(1:6, nrow=2, dimnames=list(c('a','b'),c('c','d','this that')))
latexTabular(x) # a character string with LaTeX markup
```

---

 latexTherm

---

*Create LaTeX Thermometers and Colored Needles*


---

**Description**

latexTherm creates a LaTeX picture environment for drawing a series of thermometers whose heights depict the values of a variable y assumed to be scaled from 0 to 1. This is useful for showing fractions of sample analyzed in any table or plot, intended for a legend. For example, four thermometers might be used to depict the fraction of enrolled patients included in the current analysis, the fraction randomized, the fraction of patients randomized to treatment A being analyzed, and the fraction randomized to B being analyzed. The picture is placed inside a LaTeX macro definition for macro variable named name, to be invoked by the user later in the LaTeX file using name preceded by a backslash.



If `y` has an attribute "table", it is assumed to contain a character string with LaTeX code. This code is used as a tooltip popup for PDF using the LaTeX `ocgtools` package or using `style tooltips`. Typically the code will contain a `tabular` environment. The user must define a LaTeX macro `tooltipn` that takes two arguments (original object and pop-up object) that does the pop-up.

`latexNeedle` is similar to `latexTherm` except that vertical needles are produced and each may have its own color. A grayscale box is placed around the needles and provides the 0-1 y-axis reference. Horizontal grayscale grid lines may be drawn.

`pngNeedle` is similar to `latexNeedle` but is for generating small png graphics. The full graphics file name is returned invisibly.

### Usage

```
latexTherm(y, name, w = 0.075, h = 0.15, spacefactor = 1/2, extra = 0.07,
           file = "", append = TRUE)
```

```
latexNeedle(y, x=NULL, col='black', href=0.5, name, w=.05, h=.15,
            extra=0, file = "", append=TRUE)
```

```
pngNeedle(y, x=NULL, col='black', href=0.5, lwd=3.5, w=6, h=18,
           file=tempfile(fileext='.png'))
```

### Arguments

<code>y</code>	a vector of 0-1 scaled values. Boxes and their frames are omitted for NA elements
<code>x</code>	a vector corresponding to <code>y</code> giving x-coordinates. Scaled accordingly, or defaults to equally-spaced values.
<code>name</code>	name of LaTeX macro variable to be defined
<code>w</code>	width of a single box (thermometer) in inches. For <code>latexNeedle</code> and <code>pngNeedle</code> is the spacing between needles, the latter being in pixels.
<code>h</code>	height of a single box in inches. For <code>latexNeedle</code> and <code>pngNeedle</code> is the height of the frame, the latter in pixels.
<code>spacefactor</code>	fraction of <code>w</code> added for extra space between boxes for <code>latexTherm</code>
<code>extra</code>	extra space in inches to set aside to the right of and above the series of boxes or frame
<code>file</code>	name of file to which to write LaTeX code. Default is the console. Also used as base file name for png graphic. Default for that is from <code>tempfile</code> .
<code>append</code>	set to FALSE to write over <code>file</code>
<code>col</code>	a vector of colors corresponding to positions in <code>y</code> . <code>col</code> is repeated if too short.
<code>href</code>	values of <code>y</code> (0-1) for which horizontal grayscale reference lines are drawn for <code>latexNeedle</code> and <code>pngNeedle</code> . Set to NULL to not draw any reference lines
<code>lwd</code>	line width of needles for <code>pngNeedle</code>

### Author(s)

Frank Harrell

**Examples**

```
## Not run:
# The following is in the Hmisc tests directory
# For a knitr example see latexTherm.Rnw in that directory
ct <- function(...) cat(..., sep='')
ct('\documentclass{report}\begin{document}\n')
latexTherm(c(1, 1, 1, 1), name='lta')
latexTherm(c(.5, .7, .4, .2), name='ltb')
latexTherm(c(.5, NA, .75, 0), w=.3, h=1, name='ltc', extra=0)
latexTherm(c(.5, NA, .75, 0), w=.3, h=1, name='ltcc')
latexTherm(c(0, 0, 0, 0), name='ltd')
ct('This is a the first:\lta and the second:\ltb\\ and the third
without extra:\ltc END\\nThird with extra:\ltcc END\\
\vspace{2in}\\
All data = zero, frame only:\ltd\\
\end{document}\n')
w <- pngNeedle(c(.2, .5, .7))
cat(tobase64image(w)) # can insert this directly into an html file

## End(Not run)
```

---

legendfunctions

*Legend Creation Functions*


---

**Description**

Wrappers to plot defined legend plotting functions

**Usage**

```
Key(...)
Key2(...)
sKey(...)
```

**Arguments**

... arguments to pass to wrapped functions

---

list.tree

*Pretty-print the Structure of a Data Object*


---

**Description**

This is a function to pretty-print the structure of any data object (usually a list). It is similar to the R function `str`.

**Usage**

```
list.tree(struct, depth=-1, numbers=FALSE, maxlen=22, maxcomp=12,  
          attr.print=TRUE, front="", fill=".", name.of, size=TRUE)
```

**Arguments**

struct	The object to be displayed
depth	Maximum depth of recursion (of lists within lists ...) to be printed; negative value means no limit on depth.
numbers	If TRUE, use numbers in leader instead of dots to represent position in structure.
maxlen	Approximate maximum length (in characters) allowed on each line to give the first few values of a vector. maxlen=0 suppresses printing any values.
maxcomp	Maximum number of components of any list that will be described.
attr.print	Logical flag, determining whether a description of attributes will be printed.
front	Front material of a line, for internal use.
fill	Fill character used for each level of indentation.
name.of	Name of object, for internal use (deparsed version of struct by default).
size	Logical flag, should the size of the object in bytes be printed?  A description of the structure of struct will be printed in outline form, with indentation for each level of recursion, showing the internal storage mode, length, class(es) if any, attributes, and first few elements of each data vector. By default each level of list recursion is indicated by a "." and attributes by "A".

**Author(s)**

Alan Zaslavsky, <zaslavsk@hcp.med.harvard.edu>

**See Also**

[str](#)

**Examples**

```
X <- list(a=ordered(c(1:30,30:1)),b=c("Rick","John","Allan"),  
         c=diag(300),e=cbind(p=1008:1019,q=4))  
list.tree(X)  
# In R you can say str(X)
```

makeNstr *creates a string that is a repeat of a substring*

---

**Description**

Takes a character and creates a string that is the character repeated len times.

**Usage**

```
makeNstr(char, len)
```

**Arguments**

char	character to be repeated
len	number of times to repeat char.

**Value**

A string that is char repeated len times.

**Author(s)**

Charles Dupont

**See Also**

[paste](#), [rep](#)

**Examples**

```
makeNstr(" ", 5)
```

---

mApply *Apply a Function to Rows of a Matrix or Vector*

---

**Description**

mApply is like tapply except that the first argument can be a matrix or a vector, and the output is cleaned up if simplify=TRUE. It uses code adapted from Tony Plate (<tplate@blackmesacapital.com>) to operate on grouped submatrices.

As mApply can be much faster than using by, it is often worth the trouble of converting a data frame to a numeric matrix for processing by mApply. asNumericMatrix will do this, and matrix2dataFrame will convert a numeric matrix back into a data frame.

**Usage**

```
mApply(X, INDEX, FUN, ..., simplify=TRUE, keepmatrix=FALSE)
```

**Arguments**

X	a vector or matrix capable of being operated on by the function specified as the FUN argument
INDEX	list of factors, each of same number of rows as 'X' has.
FUN	the function to be applied. In the case of functions like '+', '
...	optional arguments to 'FUN'.
simplify	set to 'FALSE' to suppress simplification of the result in to an array, matrix, etc.
keepmatrix	set to TRUE to keep result as a matrix even if simplify is TRUE, in the case of only one stratum

**Value**

For `mApply`, the returned value is a vector, matrix, or list. If `FUN` returns more than one number, the result is an array if `simplify=TRUE` and is a list otherwise. If a matrix is returned, its rows correspond to unique combinations of `INDEX`. If `INDEX` is a list with more than one vector, `FUN` returns more than one number, and `simplify=FALSE`, the returned value is a list that is an array with the first dimension corresponding to the last vector in `INDEX`, the second dimension corresponding to the next to last vector in `INDEX`, etc., and the elements of the list-array correspond to the values computed by `FUN`. In this situation the returned value is a regular array if `simplify=TRUE`. The order of dimensions is as previously but the additional (last) dimension corresponds to values computed by `FUN`.

**Author(s)**

Frank Harrell  
Department of Biostatistics  
Vanderbilt University  
<fh@fharrell.com>

**See Also**

[asNumericMatrix](#), [matrix2dataFrame](#), [tapply](#), [sapply](#), [lapply](#), [mapply](#), [by](#).

**Examples**

```
require(datasets, TRUE)
a <- mApply(iris[,-5], iris$Species, mean)
```

**Description**

mChoice is a function that is useful for grouping variables that represent individual choices on a multiple choice question. These choices are typically factor or character values but may be of any type. Levels of component factor variables need not be the same; all unique levels (or unique character values) are collected over all of the multiple variables. Then a new character vector is formed with integer choice numbers separated by semicolons. Optimally, a database system would have exported the semicolon-separated character strings with a levels attribute containing strings defining value labels corresponding to the integer choice numbers. mChoice is a function for creating a multiple-choice variable after the fact. mChoice variables are explicitly handled by the describe and summary. formula functions. NAs or blanks in input variables are ignored.

format.mChoice will convert the multiple choice representation to text form by substituting levels for integer codes. as.double.mChoice converts the mChoice object to a binary numeric matrix, one column per used level (or all levels of drop=FALSE. This is called by the user by invoking as.numeric. There is a print method and a summary method, and a print method for the summary.mChoice object. The summary method computes frequencies of all two-way choice combinations, the frequencies of the top 5 combinations, information about which other choices are present when each given choice is present, and the frequency distribution of the number of choices per observation. This summary output is used in the describe function. The print method returns an html character string if options(prType='html') is in effect if render=FALSE or renders the html otherwise. This is used by print.describe and is most effective when short=TRUE is specified to summary.

in.mChoice creates a logical vector the same length as x whose elements are TRUE when the observation in x contains at least one of the codes or value labels in the second argument.

match.mChoice creates an integer vector of the indexes of all elements in table which contain any of the specified levels

nmChoice returns an integer vector of the number of choices that were made

is.mChoice returns TRUE if the argument is a multiple choice variable.

**Usage**

```
mChoice(..., label='',
         sort.levels=c('original','alphabetic'),
         add.none=FALSE, drop=TRUE, ignoreNA=TRUE)
```

```
## S3 method for class 'mChoice'
format(x, minlength=NULL, sep=";", ...)
```

```
## S3 method for class 'mChoice'
as.double(x, drop=FALSE, ...)
```

```
## S3 method for class 'mChoice'
```

```

print(x, quote=FALSE, max.levels=NULL,
      width=getOption("width"), ...)

## S3 method for class 'mChoice'
as.character(x, ...)

## S3 method for class 'mChoice'
summary(object, ncombos=5, minlength=NULL,
        drop=TRUE, short=FALSE, ...)

## S3 method for class 'summary.mChoice'
print(x, prlabel=TRUE, render=TRUE, ...)

## S3 method for class 'mChoice'
x[..., drop=FALSE]

match.mChoice(x, table, nomatch=NA, incomparables=FALSE)

inmChoice(x, values, condition=c('any', 'all'))

inmChoicelike(x, values, condition=c('any', 'all'),
             ignore.case=FALSE, fixed=FALSE)

nmChoice(object)

is.mChoice(x)

## S3 method for class 'mChoice'
Summary(..., na.rm)

```

### Arguments

na.rm	Logical: remove NA's from data
table	a vector (mChoice) of values to be matched against.
nomatch	value to return if a value for x does not exist in table.
incomparables	logical whether incomparable values should be compared.
...	a series of vectors
label	a character string label attribute to attach to the matrix created by mChoice
sort.levels	set sort.levels="alphabetic" to sort the columns of the matrix created by mChoice alphabetically by category rather than by the original order of levels in component factor variables (if there were any input variables that were factors)
add.none	Set add.none to TRUE to make a new category 'none' if it doesn't already exist and if there is an observations with no choices selected.
drop	set drop=FALSE to keep unused factor levels as columns of the matrix produced by mChoice
ignoreNA	set to FALSE to keep any NAs present in data as a real level. Prior to Hmisc 4.7-2 FALSE was the default.

x	an object of class "mchoice" such as that created by mChoice. For is.mChoice is any object.
object	an object of class "mchoice" such as that created by mChoice
ncombos	maximum number of combos.
width	Width of a line of text to be formatted
quote	quote the output
max.levels	max levels to be displayed
minlength	By default no abbreviation of levels is done in format and summary. Specify a positive integer to use abbreviation in those functions. See <a href="#">abbreviate</a> .
short	set to TRUE to have summary.mChoice use integer choice numbers in its tables, and to print the choice level definitions at the top
sep	character to use to separate levels when formatting
prlabel	set to FALSE to keep print.summary.mChoice from printing the variable label and number of unique values. Ignore for html output.
render	applies of options(prType='html') is in effect. Set to FALSE to return the html text instead of rendering the html.
values	a scalar or vector. If values is integer, it is the choice codes, and if it is a character vector, it is assumed to be value labels. For inmChoicelike values must be character strings which are pieces of choice labels.
condition	set to 'all' for inmChoice to require that all choices in values be present instead of the default of any of them present.
ignore.case	set to TRUE to have inmChoicelike ignore case in the data when matching on values
fixed	see grep

### Value

mChoice returns a character vector of class "mChoice" plus attributes "levels" and "label". summary.mChoice returns an object of class "summary.mChoice". inmChoice and inmChoicelike return a logical vector. format.mChoice returns a character vector, and as.double.mChoice returns a binary numeric matrix. nmChoice returns an integer vector. print.summary.mChoice returns an html character string if options(prType='html') is in effect.

### Author(s)

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>

### See Also

[label](#), [combplotp](#)



**Examples**

```

options(digits=3)
set.seed(3)
n <- 20
sex <- factor(sample(c("m","f"), n, rep=TRUE))
age <- rnorm(n, 50, 5)
treatment <- factor(sample(c("Drug","Placebo"), n, rep=TRUE))

# Generate a 3-choice variable; each of 3 variables has 5 possible levels
symp <- c('Headache','Stomach Ache','Hangnail',
         'Muscle Ache','Depressed')
symptom1 <- sample(symp, n, TRUE)
symptom2 <- sample(symp, n, TRUE)
symptom3 <- sample(symp, n, TRUE)
cbind(symptom1, symptom2, symptom3)[1:5,]
Symptoms <- mChoice(symptom1, symptom2, symptom3, label='Primary Symptoms')
Symptoms
print(Symptoms, long=TRUE)
format(Symptoms[1:5])
inmChoice(Symptoms,'Headache')
inmChoicelike(Symptoms, 'head', ignore.case=TRUE)
levels(Symptoms)
inmChoice(Symptoms, 3)
# Find all subjects with either of two symptoms
inmChoice(Symptoms, c('Headache','Hangnail'))
# Note: In this example, some subjects have the same symptom checked
# multiple times; in practice these redundant selections would be NAs
# mChoice will ignore these redundant selections
# Find all subjects with both symptoms
inmChoice(Symptoms, c('Headache', 'Hangnail'), condition='all')

meanage <- N <- numeric(5)
for(j in 1:5) {
  meanage[j] <- mean(age[inmChoice(Symptoms,j)])
  N[j] <- sum(inmChoice(Symptoms,j))
}
names(meanage) <- names(N) <- levels(Symptoms)
meanage
N

# Manually compute mean age for 2 symptoms
mean(age[symptom1=='Headache' | symptom2=='Headache' | symptom3=='Headache'])
mean(age[symptom1=='Hangnail' | symptom2=='Hangnail' | symptom3=='Hangnail'])

summary(Symptoms)

#Frequency table sex*treatment, sex*Symptoms
summary(sex ~ treatment + Symptoms, fun=table)
# Check:
ma <- inmChoice(Symptoms, 'Muscle Ache')
table(sex[ma])

```

```
# could also do:
# summary(sex ~ treatment + mChoice(symptom1,symptom2,symptom3), fun=table)

#Compute mean age, separately by 3 variables
summary(age ~ sex + treatment + Symptoms)

summary(age ~ sex + treatment + Symptoms, method="cross")

f <- summary(treatment ~ age + sex + Symptoms, method="reverse", test=TRUE)
f
# trio of numbers represent 25th, 50th, 75th percentile
print(f, long=TRUE)
```

---

 mdb.get

*Read Tables in a Microsoft Access Database*


---

## Description

Assuming the mdbtools package has been installed on your system and is in the system path, mdb.get imports one or more tables in a Microsoft Access database. Date-time variables are converted to dates or chron package date-time variables. The csv.get function is used to import automatically exported csv files. If tables is unspecified all tables in the database are retrieved. If more than one table is imported, the result is a list of data frames.

## Usage

```
mdb.get(file, tables=NULL, lowernames=FALSE, allow=NULL,
        dateformat='%m/%d/%y', mdbexportArgs='-b strip', ...)
```

## Arguments

file	the file name containing the Access database
tables	character vector specifying the names of tables to import. Default is to import all tables. Specify tables=TRUE to return the list of available tables.
lowernames	set this to TRUE to change variable names to lower case
allow	a vector of characters allowed by R that should not be converted to periods in variable names. By default, underscores in variable names are converted to periods as with R before version 1.9.
dateformat	see <a href="#">cleanup.import</a> . Default is the usual Access format used in the U.S.
mdbexportArgs	command line arguments to issue to mdb-export. Set to ' ' to omit '-b strip'.
...	arguments to pass to csv.get

**Details**

Uses the mdbtools package executables mdb-tables, mdb-schema, and mdb-export (with by default option -b strip to drop any binary output). In Debian/Ubuntu Linux run `apt-get install mdbtools`. `cleanup.import` is invoked by `csv.get` to transform variables and store them as efficiently as possible.

**Value**

a new data frame or a list of data frames

**Author(s)**

Frank Harrell, Vanderbilt University

**See Also**

[data.frame](#), [cleanup.import](#), [csv.get](#), [Date](#), [chron](#)

**Examples**

```
## Not run:
# Read all tables in the Microsoft Access database Nwind.mdb
d <- mdb.get('Nwind.mdb')
contents(d)
for(z in d) print(contents(z))
# Just print the names of tables in the database
mdb.get('Nwind.mdb', tables=TRUE)
# Import one table
Orders <- mdb.get('Nwind.mdb', tables='Orders')

## End(Not run)
```

---

meltData

*meltData*

---

**Description**

Melt a Dataset To Examine All Xs vs Y

**Usage**

```
meltData(
  formula,
  data,
  tall = c("right", "left"),
  vnames = c("labels", "names"),
  sepunits = FALSE,
  ...
)
```

**Arguments**

formula	a formula
data	data frame or table
tall	see above
vnames	set to names to always use variable names instead of labels for X
sepunits	set to TRUE to create a separate variable Units to hold units of measurement. The variable is not created if no original variables have a non-blank units attribute.
...	passed to label()

**Details**

Uses a formula with one or more left hand side variables (Y) and one or more right hand side variables (X). Uses `data.table::melt()` to melt data so that each X is played against the same Y if `tall='right'` (the default) or each Y is played against the same X combination if `tall='left'`. The resulting data table has variables Y with their original names (if `tall='right'`) or variables X with their original names (if `tall='left'`), variable, and value. By default variable is taken as `label()`s of the tall variables.

**Value**

data table

**Author(s)**

Frank Harrell

**See Also**

[label\(\)](#)

**Examples**

```
d <- data.frame(y1=(1:10)/10, y2=(1:10)/100, x1=1:10, x2=101:110)
label(d$x1) <- 'X1'
units(d$x1) <- 'mmHg'
m=meltData(y1 + y2 ~ x1 + x2, data=d, units=TRUE) # consider also html=TRUE
print(m)
m=meltData(y1 + y2 ~ x1 + x2, data=d, tall='left')
print(m)
```

---

Merge

---

*Merge Multiple Data Frames or Data Tables*


---

### Description

Merges an arbitrarily large series of data frames or data tables containing common `id` variables. Information about number of observations and number of unique `ids` in individual and final merged datasets is printed. The first data frame/table has special meaning in that all of its observations are kept whether they match `ids` in other data frames or not. For all other data frames, by default non-matching observations are dropped. The first data frame is also the one against which counts of unique `ids` are compared. Sometimes merge drops variable attributes such as `labels` and `units`. These are restored by Merge.

### Usage

```
Merge(..., id = NULL, all = TRUE, verbose = TRUE)
```

### Arguments

<code>...</code>	two or more dataframes or data tables
<code>id</code>	a formula containing all the identification variables such that the combination of these variables uniquely identifies subjects or records of interest. May be omitted for data tables; in that case the key function retrieves the <code>id</code> variables.
<code>all</code>	set to <code>FALSE</code> to drop observations not found in second and later data frames (only applies if not using <code>data.table</code> )
<code>verbose</code>	set to <code>FALSE</code> to not print information about observations

### Examples

```
## Not run:
a <- data.frame(sid=1:3, age=c(20,30,40))
b <- data.frame(sid=c(1,2,2), bp=c(120,130,140))
d <- data.frame(sid=c(1,3,4), wt=c(170,180,190))
all <- Merge(a, b, d, id = ~ sid)
# First file should be the master file and must
# contain all ids that ever occur. ids not in the master will
# not be merged from other datasets.
a <- data.table(a); setkey(a, sid)
# data.table also does not allow duplicates without allow.cartesian=TRUE
b <- data.table(sid=1:2, bp=c(120,130)); setkey(b, sid)
d <- data.table(d); setkey(d, sid)
all <- Merge(a, b, d)

## End(Not run)
```

mgp.axis

*Draw Axes With Side-Specific mgp Parameters***Description**

mgp.axis is a version of axis that uses the appropriate side-specific mgp parameter (see [par](#)) to account for different space requirements for axis labels vertical vs. horizontal tick marks. mgp.axis also fixes a bug in axis(2, ...) that causes it to assume las=1.

mgp.axis.labels is used so that different spacing between tick marks and axis tick mark labels may be specified for x- and y-axes. Use mgp.axis.labels('default') to set defaults. Users can set values manually using mgp.axis.labels(x,y) where x and y are 2nd value of par('mgp') to use. Use mgp.axis.labels(type=w) to retrieve values, where w='x', 'y', 'x and y', 'xy', to get 3 mgp values (first 3 types) or 2 mgp.axis.labels.

**Usage**

```
mgp.axis(side, at = NULL, ...,
         mgp = mgp.axis.labels(type = if (side == 1 | side == 3) "x"
                               else "y"),
         axistitle = NULL, cex.axis=par('cex.axis'), cex.lab=par('cex.lab'))
```

```
mgp.axis.labels(value,type=c('xy','x','y','x and y'))
```

**Arguments**

side, at	see <a href="#">par</a>
...	arguments passed through to <a href="#">axis</a>
mgp, cex.axis, cex.lab	see <a href="#">par</a>
axistitle	if specified will cause axistitle to be drawn on the appropriate axis as a title
value	vector of values to which to set system option mgp.axis.labels
type	see above

**Value**

mgp.axis.labels returns the value of mgp (only the second element of mgp if type="xy" or a list with elements x and y if type="x or y", each list element being a 3-vector) for the appropriate axis if value is not specified, otherwise it returns nothing but the system option mgp.axis.labels is set.

mgp.axis returns nothing.

**Side Effects**

mgp.axis.labels stores the value in the system option mgp.axis.labels

**Author(s)**

Frank Harrell

**See Also**[par](#)**Examples**

```
## Not run:
mgp.axis.labels(type='x') # get default value for x-axis
mgp.axis.labels(type='y') # get value for y-axis
mgp.axis.labels(type='xy') # get 2nd element of both mgps
mgp.axis.labels(type='x and y') # get a list with 2 elements
mgp.axis.labels(c(3,.5,0), type='x') # set
options('mgp.axis.labels') # retrieve

plot(..., axes=FALSE)
mgp.axis(1, "X Label")
mgp.axis(2, "Y Label")

## End(Not run)
```

---

 mhgr

---

*Miscellaneous Functions for Epidemiology*


---

**Description**

The `mhgr` function computes the Cochran-Mantel-Haenszel stratified risk ratio and its confidence limits using the Greenland-Robins variance estimator.

The `lrcum` function takes the results of a series of 2x2 tables representing the relationship between test positivity and diagnosis and computes positive and negative likelihood ratios (with all their deficiencies) and the variance of their logarithms. Cumulative likelihood ratios and their confidence intervals (assuming independence of tests) are computed, assuming a string of all positive tests or a string of all negative tests. The method of Simel et al as described in Altman et al is used.

**Usage**

```
mhgr(y, group, strata, conf.int = 0.95)
## S3 method for class 'mhgr'
print(x, ...)

lrcum(a, b, c, d, conf.int = 0.95)
## S3 method for class 'lrcum'
print(x, dec=3, ...)
```

**Arguments**

y	a binary response variable
group	a variable with two unique values specifying comparison groups
strata	the stratification variable
conf.int	confidence level
x	an object created by mhgr or lrcum
a	frequency of true positive tests
b	frequency of false positive tests
c	frequency of false negative tests
d	frequency of true negative tests
dec	number of places to the right of the decimal to print for lrcum
...	additional arguments to be passed to other print functions

**Details**

Uses equations 4 and 13 from Greenland and Robins.

**Value**

a list of class "mhgr" or of class "lrcum".

**Author(s)**

Frank E Harrell Jr <fh@fharrell.com>

**References**

Greenland S, Robins JM (1985): Estimation of a common effect parameter from sparse follow-up data. *Biometrics* 41:55-68.

Altman DG, Machin D, Bryant TN, Gardner MJ, Eds. (2000): *Statistics with Confidence*, 2nd Ed. Bristol: BMJ Books, 105-110.

Simel DL, Samsa GP, Matchar DB (1991): Likelihood ratios with confidence: sample size estimation for diagnostic test studies. *J Clin Epi* 44:763-770.

**See Also**

[logrank](#)

**Examples**

```
# Create Migraine dataset used in Example 28.6 in the SAS PROC FREQ guide
d <- expand.grid(response=c('Better','Same'),
                 treatment=c('Active','Placebo'),
                 sex=c('female','male'))
d$count <- c(16, 11, 5, 20, 12, 16, 7, 19)
d
```



```

# Expand data frame to represent raw data
r <- rep(1:8, d$count)
d <- d[r,]
with(d, mhgr(response=='Better', treatment, sex))

# Discrete survival time example, to get Cox-Mantel relative risk and CL
# From Stokes ME, Davis CS, Koch GG, Categorical Data Analysis Using the
# SAS System, 2nd Edition, Sectino 17.3, p. 596-599
#
# Input data in Table 17.5
d <- expand.grid(treatment=c('A','P'), center=1:3)
d$healed2w <- c(15,15,17,12, 7, 3)
d$healed4w <- c(17,17,17,13,17,17)
d$notHealed4w <- c( 2, 7,10,15,16,18)
d
# Reformat to the way most people would collect raw data
d1 <- d[rep(1:6, d$healed2w),]
d1$time <- '2'
d1$y <- 1
d2 <- d[rep(1:6, d$healed4w),]
d2$time <- '4'
d2$y <- 1
d3 <- d[rep(1:6, d$notHealed4w),]
d3$time <- '4'
d3$y <- 0
d <- rbind(d1, d2, d3)
d$healed2w <- d$healed4w <- d$notHealed4w <- NULL
d
# Finally, duplicate appropriate observations to create 2 and 4-week
# risk sets. Healed and not healed at 4w need to be in the 2-week
# risk set as not healed
d2w <- subset(d, time=='4')
d2w$time <- '2'
d2w$y <- 0
d24 <- rbind(d, d2w)
with(d24, table(y, treatment, time, center))
# Matches Table 17.6

with(d24, mhgr(y, treatment, interaction(center, time, sep=';')))

# Get cumulative likelihood ratios and their 0.95 confidence intervals
# based on the following two tables
#
#           Disease           Disease
#           +     -           +     -
# Test +    39    3           20    5
# Test -    21   17           22   15

lrcum(c(39,20), c(3,5), c(21,22), c(17,15))

```

**Description**

Adds minor tick marks to an existing plot. All minor tick marks that will fit on the axes will be drawn.

**Usage**

```
minor.tick(nx=2, ny=2, tick.ratio=0.5, x.args = list(), y.args = list())
```

**Arguments**

<code>nx</code>	number of intervals in which to divide the area between major tick marks on the X-axis. Set to 1 to suppress minor tick marks.
<code>ny</code>	same as <code>nx</code> but for the Y-axis.
<code>tick.ratio</code>	ratio of lengths of minor tick marks to major tick marks. The length of major tick marks is retrieved from <code>par("tck")</code> .
<code>x.args</code>	additional arguments (e.g. <code>post</code> , <code>lwd</code> ) used by <code>axis()</code> function when rendering the X-axis.
<code>y.args</code>	same as <code>x.args</code> but for Y-axis.

**Side Effects**

plots

**Author(s)**

Frank Harrell  
Department of Biostatistics  
Vanderbilt University  
<fh@fharrell.com>  
Earl Bellinger  
Max Planck Institute  
<earlbellingergmail.com>  
Viktor Horvath  
Brandeis University  
<vhorvath@brandeis.edu>

**See Also**

[axis](#)

**Examples**

```
# Plot with default settings
plot(runif(20), runif(20))
minor.tick()

# Plot with arguments passed to axis()
plot(c(0,1), c(0,1), type = 'n', axes = FALSE, ann = FALSE)
# setting up a plot without axes and annotation
```

```

points(runif(20), runif(20))           # plotting data
axis(1, pos = 0.5, lwd = 2)           # showing X-axis at Y = 0.5 with formatting
axis(2, col = 2)                       # formatted Y-axis
minor.tick( nx = 4, ny = 4, tick.ratio = 0.3,
            x.args = list(pos = 0.5, lwd = 2), # X-minor tick format argumnets
            y.args = list(col = 2))       # Y-minor tick format arguments

```

---

Misc

*Miscellaneous Functions*


---

## Description

This documents miscellaneous small functions in Hmisc that may be of interest to users.

`clowess` runs `lowess` but if the `iter` argument exceeds zero, sometimes wild values can result, in which case `lowess` is re-run with `iter=0`.

`confbar` draws multi-level confidence bars using small rectangles that may be of different colors.

`getLatestSource` fetches and sources the most recent source code for functions in GitHub.

`grType` retrieves the system option `grType`, which is forced to be "base" if the `plotly` package is not installed.

`prType` retrieves the system option `prType`, which is set to "plain" if the option is not set. `print` methods that allow for markdown/html/latex can be automatically invoked by setting `options(prType="html")` or `options(prType='latex')`.

`htmlSpecialType` retrieves the system option `htmlSpecialType`, which is set to "unicode" if the option is not set. `htmlSpecialType='unicode'` cause html-generating functions in Hmisc and rms to use unicode for special characters, and `htmlSpecialType='&'` uses the older ampersand 3-digit format.

`inverseFunction` generates a function to find all inverses of a monotonic or nonmonotonic function that is tabulated at vectors (x,y), typically 1000 points. If the original function is monotonic, simple linear interpolation is used and the result is a vector, otherwise linear interpolation is used within each interval in which the function is monotonic and the result is a matrix with number of columns equal to the number of monotonic intervals. If a requested y is not within any interval, the extreme x that pertains to the nearest extreme y is returned. Specifying `what='sample'` to the returned function will cause a vector to be returned instead of a matrix, with elements taken as a random choice of the possible inverses.

`james.stein` computes James-Stein shrunk estimates of cell means given a response variable (which may be binary) and a grouping indicator.

`keepHattrib` for an input variable or a data frame, creates a list object saving special Hmisc attributes such as `label` and `units` that might be lost during certain operations such as running `data.table`. `restoreHattrib` restores these attributes.

`km.quick` provides a fast way to invoke `survfitKM` in the `survival` package to get Kaplan-Meier estimates for a single stratum for a vector of time points (if `times` is given) or to get a vector of survival time quantiles (if `q` is given).

`latexBuild` takes pairs of character strings and produces a single character string containing concatenation of all of them, plus an attribute "close" which is a character string containing the LaTeX

closure that will balance LaTeX code with respect to parentheses, braces, brackets, or `begin` vs. `end`. When an even-numbered element of the vector is not a left parenthesis, brace, or bracket, the element is taken as a word that was surrounded by `begin` and braces, for which the corresponding end is constructed in the returned attribute.

`lm.fit.qr.bare` is a fast stripped-down function for computing regression coefficients, residuals,  $R^2$ , and fitted values. It uses `lm.fit`.

`matxv` multiplies a matrix by a vector, handling automatic addition of intercepts if the matrix does not have a column of ones. If the first argument is not a matrix, it will be converted to one. An optional argument allows the second argument to be treated as a matrix, useful when its rows represent bootstrap reps of coefficients. Then  $ab'$  is computed. `matxv` respects the "intercepts" attribute if it is stored on `b` by the `rms` package. This is used by `orm` fits that are bootstrap-repeated by `bootcov` where only the intercept corresponding to the median is retained. If `kint` has nonzero length, it is checked for consistency with the attribute.

`makeSteps` is a copy of the `dostep` function inside the `survival` package's `plot.survfit` function. It expands a series of points to include all the segments needed to plot step functions. This is useful for drawing polygons to shade confidence bands for step functions.

`nomiss` returns a data frame (if its argument is one) with rows corresponding to NAs removed, or it returns a matrix with rows with any element missing removed.

`outerText` uses `axis()` to put right-justified text strings in the right margin. Placement depends on `par('mar')[4]`

`plotlyParm` is a list of functions useful for specifying parameters to `plotly` graphics.

`plotp` is a generic to handle `plotp` methods to make `plotly` graphics.

`rendHTML` renders HTML in a character vector, first converting to one character string with newline delimiters. If `knitr` is currently running, runs this string through `knitr::asis_output` so that the user need not include `results='asis'` in the chunk header for R Markdown or Quarto. If `knitr` is not running, uses `htmltools::browsable` and `htmltools::HTML` and prints the result so that an RStudio viewer (if running inside RStudio) or separate browser window displays the rendered HTML. The HTML code is surrounded by `yaml` markup to make `Pandoc` not fiddle with the HTML. Set the argument `html=FALSE` to not add this, in case you are really rendering markdown. `html=FALSE` also invokes `rmarkdown::render` to convert the character vector to HTML before using `htmltools` to view, assuming the characters represent R Markdown/Quarto text other than the `YAML` header. If `options(rawmarkup=TRUE)` is in effect, `rendHTML` will just `cat()` its first argument. This is useful when rendering is happening inside a Quarto margin, for example.

`sepUnitsTrans` converts character vectors containing values such as `c("3 days", "3day", "4month", "2 years", "2weeks", "7")` to numeric vectors (here `c(3, 3, 122, 730, 14, 7)`) in a flexible fashion. The user can specify a vector of units of measurements and conversion factors. The units with a conversion factor of 1 are taken as the target units, and if those units are present in the character strings they are ignored. The target units are added to the resulting vector as the "units" attribute.

`strgraphwrap` is like `strwrap` but is for the current graphics environment.

`tobase64image` is a function written by Dirk Eddelbuettel that uses the `base64enc` package to convert a `png` graphic file to base64 encoding to include as an inline image in an `html` file.

`trap.rule` computes the area under a curve using the trapezoidal rule, assuming `x` is sorted.

`trellis.strip.blank` sets up `Trellis` or `Lattice` graphs to have a clear background on the strips for panel labels.

unPaste provides a version of the S-Plus unpaste that works for R and S-Plus.

whichClosePW is a very fast function using weighted multinomial sampling to determine which element of a vector is "closest" to each element of another vector. whichClosest quickly finds the closest element without any randomness.

whichClosek is a slow function that finds, after jittering the lookup table, the k closest matches to each element of the other vector, and chooses from among these one at random.

xless is a function for Linux/Unix users to invoke the system xless command to pop up a window to display the result of printing an object. For MacOS xless uses the system open command to pop up a TextEdit window.

## Usage

```

confbar(at, est, se, width, q = c(0.7, 0.8, 0.9, 0.95, 0.99),
        col = gray(c(0, 0.25, 0.5, 0.75, 1)),
        type = c("v", "h"), labels = TRUE, ticks = FALSE,
        cex = 0.5, side = "l", lwd = 5, clip = c(-1e+30, 1e+30),
        fun = function(x) x,
        qfun = function(x) ifelse(x == 0.5, qnorm(x),
                                ifelse(x < 0.5, qnorm(x/2),
                                        qnorm((1 + x)/2))))
getLatestSource(x=NULL, package='Hmisc', recent=NULL, avail=FALSE)
grType()
prType()
htmlSpecialType()
inverseFunction(x, y)
james.stein(y, group)
keepHattrib(obj)
km.quick(S, times, q)
latexBuild(..., insert, sep='')
lm.fit.qr.bare(x, y, tolerance, intercept=TRUE, xpqi=FALSE, singzero=FALSE)
matxv(a, b, kint=1, bmat=FALSE)
nomiss(x)
outerText(string, y, cex=par('cex'), ...)
plotlyParm
plotp(data, ...)
rendHTML(x, html=TRUE)
restoreHattrib(obj, attribs)
sepUnitsTrans(x, conversion=c(day=1, month=365.25/12, year=365.25, week=7),
              round=FALSE, digits=0)
strgraphwrap(x, width = 0.9 * getOption("width"),
             indent = 0, exdent = 0,
             prefix = "", simplify = TRUE, units='user', cex=NULL)
tobase64image(file, Rd = FALSE, alt = "image")
trap.rule(x, y)
trellis.strip.blank()
unPaste(str, sep="/"")
whichClosest(x, w)
whichClosePW(x, w, f=0.2)

```

whichClosek(x, w, k)  
 xless(x, ..., title)

### Arguments

a	a numeric matrix or vector
alt, Rd	see <code>base64::img</code>
at	x-coordinate for vertical confidence intervals, y-coordinate for horizontal
attribs	an object returned by <code>keepHattrib</code>
avail	set to TRUE to have <code>getLatestSource</code> return a data frame of available files and latest versions instead of fetching any
b	a numeric vector
cex	character expansion factor
clip	interval to truncate limits
col	vector of colors
conversion	a named numeric vector
data	an object having a <code>plotp</code> method
digits	number of digits used for round
est	vector of point estimates for confidence limits
f	a scaling constant
file	a file name
fun	function to transform scale
group	a categorical grouping variable
html	set to FALSE to tell <code>rendHTML</code> to not surround HTML code with <code>yaml</code>
insert	a list of 3-element lists for <code>latexBuild</code> . The first of each 3-element list is a character string with an environment name. The second specifies the order: "before" or "after", the former indicating that when the environment is found, the third element of the list is inserted before or after it, according to the second element.
intercept	set to FALSE to not automatically add a column of ones to the x matrix
k	get the k closest matches
kint	which element of b to add to the result if a does not contain a column for intercepts
bmat	set to TRUE to consider b a matrix of repeated coefficients, usually resampled estimates with rows corresponding to resamples
labels	set to FALSE to omit drawing confidence coefficients
lwd	line widths
package	name of package for <code>getLatestSource</code> , default is 'Hmisc'
obj	a variable, data frame, or data table
q	vector of confidence coefficients or quantiles

qfun	quantiles on transformed scale
recent	an integer telling <code>getLatestSource</code> to get the recent most recently modified files from the package
round	set to TRUE to round converted values
S	a <a href="#">Surv</a> object
se	vector of standard errors
sep	a single character string specifying the delimiter. For <code>latexBuild</code> the default is "".
side	for <code>confbar</code> is "b", "l", "t", "r" for bottom, left, top, right.
str	a character string vector
string	a character string vector
ticks	set to TRUE to draw lines between rectangles
times	a numeric vector of times
title	a character string to title a window or plot. Ignored for <code>xless</code> under MacOs.
tolerance	tolerance for judging singularity in matrix
type	"v" for vertical, "h" for horizontal.
w	a numeric vector
width	width of confidence rectangles in user units, or see <a href="#">strwrap</a>
x	a numeric vector (matrix for <code>lm.fit.qr.bare</code> ) or data frame. For <code>xless</code> may be any object that is sensible to print. For <code>sepUnitsTrans</code> is a character or factor variable. For <code>getLatestSource</code> is a character string or vector of character strings containing base file names to retrieve from CVS. Set <code>x='all'</code> to retrieve all source files. For <code>clowess</code> , <code>x</code> may also be a list with <code>x</code> and <code>y</code> components. For <code>inverseFunction</code> , <code>x</code> and <code>y</code> contain evaluations of the function whose inverse is needed. <code>x</code> is typically an equally-spaced grid of 1000 points. For <code>strgraphwrap</code> is a character vector. For <code>rendHTML</code> <code>x</code> is a character vector.
xpxi	set to TRUE to add an element to the result containing the inverse of $X'X$
singzero	set to TRUE to set coefficients corresponding to singular variables to zero instead of NA.
y	a numeric vector. For <code>inverseFunction</code> <code>y</code> is the evaluated function values at <code>x</code> .
indent, exdent, prefix	see <a href="#">strwrap</a>
simplify	see <a href="#">sapply</a>
units	see <a href="#">par</a>
...	arguments passed through to another function. For <code>latexBuild</code> represents pairs, with odd numbered elements being character strings containing LaTeX code or a zero-length object to ignore, and even-numbered elements representing LaTeX left parenthesis, left brace, or left bracket, or environment name.

**Author(s)**

Frank Harrell and Charles Dupont

**Examples**

```

trap.rule(1:100,1:100)

unPaste(c('a;b or c','ab;d','qr;s'), ';')

sepUnitsTrans(c('3 days','4 months','2 years','7'))

set.seed(1)
whichClosest(1:100, 3:5)
whichClosest(1:100, rep(3,20))

whichClosePW(1:100, rep(3,20))
whichClosePW(1:100, rep(3,20), f=.05)
whichClosePW(1:100, rep(3,20), f=1e-10)

x <- seq(-1, 1, by=.01)
y <- x^2
h <- inverseFunction(x,y)
formals(h)$turns # vertex
a <- seq(0, 1, by=.01)
plot(0, 0, type='n', xlim=c(-.5,1.5))
lines(a, h(a)[,1]) ## first inverse
lines(a, h(a)[,2], col='red') ## second inverse
a <- c(-.1, 1.01, 1.1, 1.2)
points(a, h(a)[,1])

d <- data.frame(x=1:2, y=3:4, z=5:6)
d <- upData(d, labels=c(x='X', z='Z lab'), units=c(z='mm'))
a <- keepHattrib(d)

d <- data.frame(x=1:2, y=3:4, z=5:6)
d2 <- restoreHattrib(d, a)
sapply(d2, attributes)

## Not run:
getLatestSource(recent=5) # source() most recent 5 revised files in Hmisc
getLatestSource('cut2') # fetch and source latest cut2.s
getLatestSource('all') # get everything
getLatestSource(avail=TRUE) # list available files and latest versions

## End(Not run)

```

---

 movStats

*movStats*


---

**Description**

Moving Estimates Using Overlapping Windows



**Usage**

```

movStats(
  formula,
  stat = NULL,
  discrete = FALSE,
  space = c("n", "x"),
  eps = if (space == "n") 15,
  varyeps = FALSE,
  nignore = 10,
  xinc = NULL,
  xlim = NULL,
  times = NULL,
  tunits = "year",
  msmooth = c("smoothed", "raw", "both"),
  tsmooth = c("supsmu", "lowess"),
  bass = 8,
  span = 1/4,
  maxdim = 6,
  penalty = NULL,
  trans = function(x) x,
  itrans = function(x) x,
  loess = FALSE,
  ols = FALSE,
  qreg = FALSE,
  lrm = FALSE,
  orm = FALSE,
  hare = FALSE,
  lrm_args = NULL,
  family = "logistic",
  k = 5,
  tau = (1:3)/4,
  melt = FALSE,
  data = environment(formula),
  pr = c("none", "kable", "plain", "margin")
)

```

**Arguments**

formula	a formula with the analysis variable on the left and the x-variable on the right, following by optional stratification variables
stat	function of one argument that returns a named list of computed values. Defaults to computing mean and quartiles + N except when y is binary in which case it computes moving proportions. If y has two columns the default statistics are Kaplan-Meier estimates of cumulative incidence at a vector of times.
discrete	set to TRUE if x-axis variable is discrete and no intervals should be created for windows
space	defines whether intervals used fixed width or fixed sample size

eps	tolerance for window (half width of window). For space='x' is in data units, otherwise is the sample size for half the window, not counting the middle target point.
varyeps	applies to space='n' and causes a smaller eps to be used in strata with fewer than “ observations so as to arrive at three x points
nignore	see description, default is to exclude nignore=10 points on the left and right tails from estimation and plotting
xinc	increment in x to evaluate stats, default is xlim range/100 for space='x'. For space='n' xinc defaults to m observations, where $m = \max(n/200, 1)$ .
xlim	2-vector of limits to evaluate if space='x' (default is nignore smallest to nignore largest)
times	vector of times for evaluating one minus Kaplan-Meier estimates
tunits	time units when times is given
msmooth	set to 'smoothed' or 'both' to compute lowess-smooth moving estimates. msmooth='both' will display both. 'raw' will display only the moving statistics. msmooth='smoothed' (the default) will display only the smoothed moving estimates.
tsmooth	defaults to the super-smoother 'supsmu' for after-moving smoothing. Use tsmooth='lowess' to instead use lowess.
bass	the supsmu bass parameter used to smooth the moving statistics if tsmooth='supsmu'. The default of 8 represents quite heavy smoothing.
span	the lowess span used to smooth the moving statistics
maxdim	passed to hare, default is 6
penalty	passed to hare, default is to use BIC. Specify 2 to use AIC.
trans	transformation to apply to x
itrans	inverse transformation
loess	set to TRUE to also compute loess estimates
ols	set to TRUE to include rcspline estimate of mean using ols
qreg	set to TRUE to include quantile regression estimates w rcspline
lrm	set to TRUE to include logistic regression estimates w rcspline
orm	set to TRUE to include ordinal logistic regression estimates w rcspline (mean + quantiles in tau)
hare	set to TRUE to include hazard regression estimates of incidence at times, using the polyspline package
lrm_args	a list of optional arguments to pass to lrm when lrm=TRUE, e.g., list(maxit=20)
family	link function for ordinal regression (see rms::orm)
k	number of knots to use for ols and/or qreg rcspline
tau	quantile numbers to estimate with quantile regression
melt	set to TRUE to melt data table and derive Type and Statistic
data	data.table or data.frame, default is calling frame
pr	defaults to no printing of window information. Use pr='plain' to print in the ordinary way, pr='kable' to convert the object to knitr::kable and print, or pr='margin' to convert to kable and place in the Quarto right margin. For the latter two results='asis' must be in the chunk header.

**Details**

Function to compute moving averages and other statistics as a function of a continuous variable, possibly stratified by other variables. Estimates are made by creating overlapping moving windows and computing the statistics defined in the stat function for each window. The default method, space='n' creates varying-width intervals each having a sample size of  $2 \times \text{eps} + 1$ , and the smooth estimates are made every xinc observations. Outer intervals are not symmetric in sample size (but the mean x in those intervals will reflect that) unless eps=nignore, as outer intervals are centered at observations nignore and  $n - \text{nignore} + 1$  where the default for nignore is 10. The mean x-variable within each windows is taken to represent that window. If trans and itrans are given, x means are computed on the trans(x) scale and then itrans'd. For space='x', by default estimates are made on to the nignore smallest to the nignore largest observed values of the x variable to avoid extrapolation and to help getting the moving statistics off on an adequate start for the left tail. Also by default the moving estimates are smoothed using supsmu. When melt=TRUE you can feed the result into ggplot like this: `ggplot(w, aes(x=age, y=crea, col=Type)) + geom_line() + facet_wrap(~Statistic)`

See [here](#) for several examples.

**Value**

a data table, with attribute infon which is a data frame with rows corresponding to strata and columns N, Wmean, Wmin, Wmax if stat computed N. These summarize the number of observations used in the windows. If varyeps=TRUE there is an additional column eps with the computed per-stratum eps. When space='n' and xinc is not given, the computed xinc also appears as a column. An additional attribute info is a kable object ready for printing to describe the window characteristics.

**Author(s)**

Frank Harrell

---

mtitle	<i>Margin Titles</i>
--------	----------------------

---

**Description**

Writes overall titles and subtitles after a multiple image plot is drawn. If `par()$oma==c(0,0,0,0)`, title is used instead of mtext, to draw titles or subtitles that are inside the plotting region for a single plot.

**Usage**

```
mtitle(main, ll, lc,
        lr=format(Sys.time(), '%d%b%y'),
        cex.m=1.75, cex.l=.5, ...)
```

**Arguments**

main	main title to be centered over entire figure, default is none
ll	subtitle for lower left of figure, default is none
lc	subtitle for lower center of figure, default is none
lr	subtitle for lower right of figure, default is today's date in format 23Jan91 for UNIX or R (Thu May 30 09:08:13 1996 format for Windows). Set to "" to suppress lower right title.
cex.m	character size for main, default is 1.75
cex.l	character size for subtitles
...	other arguments passed to mtext

**Value**

nothing

**Side Effects**

plots

**Author(s)**

Frank Harrell  
Department of Biostatistics, Vanderbilt University  
<fh@fharrell.com>

**See Also**

[par](#), [mtext](#), [title](#), [unix](#), [pstamp](#)

**Examples**

```
#Set up for 1 plot on figure, give a main title,  
#use date for lr  
plot(runif(20),runif(20))  
mtitle("Main Title")
```

```
#Set up for 2 x 2 matrix of plots with a lower left subtitle and overall title  
par(mfrow=c(2,2), oma=c(3,0,3,0))  
plot(runif(20),runif(20))  
plot(rnorm(20),rnorm(20))  
plot(exp(rnorm(20)),exp(rnorm(20)))  
mtitle("Main Title",ll="n=20")
```

---

multLines

*Plot Multiple Lines*


---

### Description

Plots multiple lines based on a vector `x` and a matrix `y`, draws thin vertical lines connecting limits represented by columns of `y` beyond the first. It is assumed that either (1) the second and third columns of `y` represent lower and upper confidence limits, or that (2) there is an even number of columns beyond the first and these represent ascending quantiles that are symmetrically arranged around 0.5. If options(`grType='plotly'`) is in effect, uses `plotly` graphics instead of `grid` or `base` graphics. For `plotly` you may want to set the list of possible colors, etc. using `pobj=plot_ly(colors=...)`. `lwd,lty,lwd.vert` are ignored under `plotly`.

### Usage

```
multLines(x, y, pos = c('left', 'right'), col='gray',
          lwd=1, lty=1, lwd.vert = .85, lty.vert = 1,
          alpha = 0.4, grid = FALSE,
          pobj=plotly::plot_ly(), xlim, name=colnames(y)[1], legendgroup=name,
          showlegend=TRUE, ...)
```

### Arguments

<code>x</code>	a numeric vector
<code>y</code>	a numeric matrix with number of rows equal to the number of <code>x</code> elements
<code>pos</code>	when <code>pos='left'</code> the vertical lines are drawn, right to left, to the left of the point $(x, y[,1])$ . Otherwise lines are drawn left to right to the right of the point.
<code>col</code>	a color used to connect $(x, y[,1])$ pairs. The same color but with transparency given by the <code>alpha</code> argument is used to draw the vertical lines
<code>lwd</code>	line width for main lines
<code>lty</code>	line types for main lines
<code>lwd.vert</code>	line width for vertical lines
<code>lty.vert</code>	line type for vertical lines
<code>alpha</code>	transparency
<code>grid</code>	set to <code>TRUE</code> when using <code>grid/lattice</code>
<code>pobj</code>	an already started <code>plotly</code> object to add to
<code>xlim</code>	global x-axis limits (required if using <code>plotly</code> )
<code>name</code>	trace name if using <code>plotly</code>
<code>legendgroup</code>	legend group name if using <code>plotly</code>
<code>showlegend</code>	whether or not to show traces in legend, if using <code>plotly</code>
<code>...</code>	passed to <code>add_lines</code> or <code>add_segments</code> if using <code>plotly</code>

**Author(s)**

Frank Harrell

**Examples**

```
if (requireNamespace("plotly")) {  
  x <- 1:4  
  y <- cbind(x, x-3, x-2, x-1, x+1, x+2, x+3)  
  plot(NA, NA, xlim=c(1,4), ylim=c(-2, 7))  
  multLines(x, y, col='blue')  
  multLines(x, y, col='red', pos='right')  
}
```

---

`na.delete`*Row-wise Deletion na.action*

---

**Description**

Does row-wise deletion as `na.omit`, but adds frequency of missing values for each predictor to the "na.action" attribute of the returned model frame. Optionally stores further details if `options(na.detail.response=TRUE)`

**Usage**

```
na.delete(frame)
```

**Arguments**

`frame` a model frame

**Value**

a model frame with rows deleted and the "na.action" attribute added.

**Author(s)**

Frank Harrell  
Department of Biostatistics  
Vanderbilt University  
<fh@fharrell.com>

**See Also**

[na.omit](#), [na.keep](#), [na.detail.response](#), [model.frame.default](#), [naresid](#), [naprint](#)

**Examples**

```
# options(na.action="na.delete")  
# ols(y ~ x)
```

---

na.detail.response      *Detailed Response Variable Information*


---

### Description

This function is called by certain `na.action` functions if `options(na.detail.response=TRUE)` is set. By default, this function returns a matrix of counts of non-NAs and the mean of the response variable computed separately by whether or not each predictor is NA. The default action uses the last column of a `Surv` object, in effect computing the proportion of events. Other summary functions may be specified by using `options(na.fun.response="name of function")`.

### Usage

```
na.detail.response(mf)
```

### Arguments

`mf`                    a model frame

### Value

a matrix, with rows representing the different statistics that are computed for the response, and columns representing the different subsets for each predictor (NA and non-NA value subsets).

### Author(s)

Frank Harrell  
Department of Biostatistics  
Vanderbilt University  
<fh@fharrell.com>

### See Also

[na.omit](#), [na.delete](#), [model.frame.default](#), [naresid](#), [naprint](#), [describe](#)

### Examples

```
# sex
# [1] m f f m f f m m m m m m m f f f m f m
# age
# [1] NA 41 23 30 44 22 NA 32 37 34 38 36 36 50 40 43 34 22 42 30
# y
# [1] 0 1 0 0 1 0 1 0 0 1 1 1 0 0 1 1 0 1 0 0
# options(na.detail.response=TRUE, na.action="na.delete", digits=3)
# lrm(y ~ age*sex)
#
# Logistic Regression Model
#
# lrm(formula = y ~ age * sex)
```

```

#
#
# Frequencies of Responses
#  0 1
# 10 8
#
# Frequencies of Missing Values Due to Each Variable
#  y age sex
#  0  2  0
#
#
# Statistics on Response by Missing/Non-Missing Status of Predictors
#
#   age=NA age!=NA sex!=NA Any NA  No NA
#  N     2.0 18.000  20.00   2.0 18.000
# Mean   0.5  0.444   0.45   0.5  0.444
#
# \dots\dots
# options(na.action="na.keep")
# describe(y ~ age*sex)
# Statistics on Response by Missing/Non-Missing Status of Predictors
#
#   age=NA age!=NA sex!=NA Any NA  No NA
#  N     2.0 18.000  20.00   2.0 18.000
# Mean   0.5  0.444   0.45   0.5  0.444
#
# \dots
# options(na.fun.response="table") #built-in function table()
# describe(y ~ age*sex)
#
# Statistics on Response by Missing/Non-Missing Status of Predictors
#
#   age=NA age!=NA sex!=NA Any NA  No NA
#  0     1     10     11     1     10
#  1     1     8      9     1     8
#
# \dots

```

---

na.keep

*Do-nothing na.action*


---

## Description

Does not delete rows containing NAs, but does add details concerning the distribution of the response variable if `options(na.detail.response=TRUE)`. This `na.action` is primarily for use with `describe.formula`.

## Usage

```
na.keep(mf)
```



**Arguments**

mf                    a model frame

**Value**

the same model frame with the "na.action" attribute

**Author(s)**

Frank Harrell  
Department of Biostatistics  
Vanderbilt University  
<fh@fharrell.com>

**See Also**

[na.omit](#), [na.delete](#), [model.frame.default](#), [na.detail.response](#), [naresid](#), [naprint](#), [describe](#)

**Examples**

```
options(na.action="na.keep", na.detail.response=TRUE)
x1 <- runif(20)
x2 <- runif(20)
x2[1:4] <- NA
y <- rnorm(20)
describe(y ~ x1*x2)
```

---

nCoincident

*nCoincident*

---

**Description**

Number of Coincident Points

**Usage**

```
nCoincident(x, y, bins = 400)
```

**Arguments**

x                    numeric vector  
y                    numeric vector  
bins                 number of bins in both directions

**Details**

Computes the number of x,y pairs that are likely to be obscured in a regular scatterplot, in the sense of overlapping pairs after binning into bins x bins squares where bins defaults to 400. NAs are removed first.

**Value**

integer count

**Author(s)**

Frank Harrell

**Examples**

```
nCoincident(c(1:5, 4:5), c(1:5, 4:5)/10)
```

---

nobsY

---

*Compute Number of Observations for Left Hand Side of Formula*


---

**Description**

After removing any artificial observations added by `addMarginal`, computes the number of non-missing observations for all left-hand-side variables in `formula`. If `formula` contains a term `id(variable)` `variable` is assumed to be a subject ID variable, and only unique subject IDs are counted. If `group` is given and its value is the name of a variable in the right-hand-side of the model, an additional object `nobsg` is returned that is a matrix with as many columns as there are left-hand variables, and as many rows as there are levels to the `group` variable. This matrix has the further breakdown of unique non-missing observations by `group`. The concatenation of all ID variables, is returned in a list element `id`.

**Usage**

```
nobsY(formula, group=NULL, data = NULL, subset = NULL,
      na.action = na.retain, matrixna=c('all', 'any'))
```

**Arguments**

<code>formula</code>	a formula object
<code>group</code>	character string containing optional name of a stratification variable for computing sample sizes
<code>data</code>	a data frame
<code>subset</code>	an optional subsetting criterion
<code>na.action</code>	an optional NA-handling function
<code>matrixna</code>	set to "all" if an observation is to be considered NA if all the columns of the variable are NA, otherwise use <code>matrixna="any"</code> to consider the row missing if any of the columns are missing

**Value**

an integer, with an attribute "formula" containing the original formula but with an `id` variable (if present) removed

## Examples

```
d <- expand.grid(sex=c('female', 'male', NA),
               country=c('US', 'Romania'),
               reps=1:2)
d$subject.id <- c(0, 0, 3:12)
dm <- addMarginal(d, sex, country)
dim(dm)
nobsY(sex + country ~ 1, data=d)
nobsY(sex + country ~ id(subject.id), data=d)
nobsY(sex + country ~ id(subject.id) + reps, group='reps', data=d)
nobsY(sex ~ 1, data=d)
nobsY(sex ~ 1, data=dm)
nobsY(sex ~ id(subject.id), data=dm)
```

---

nstr	<i>Creates a string of arbitrary length</i>
------	---

---

## Description

Creates a vector of strings which consists of the string segment given in each element of the string vector repeated times.

## Usage

```
nstr(string, times)
```

## Arguments

string	character: vector of string segments to be repeated. Will be recycled if argument times is longer.
times	integer: vector of number of times to repeat the corresponding segment. Will be recycled if argument string is longer.

## Value

returns a character vector the same length as the longest of the two arguments.

## Note

Will throw a warning if the length of the longer argument is not an even multiple of the shorter argument.

## Author(s)

Charles Dupont

## See Also

[paste](#), [rep](#)

**Examples**

```
nstr(c("a"), c(0,3,4))  
  
nstr(c("a", "b", "c"), c(1,2,3))  
  
nstr(c("a", "b", "c"), 4)
```

---

num.intercepts	<i>Extract number of intercepts</i>
----------------	-------------------------------------

---

**Description**

Extract the number of intercepts from a model

**Usage**

```
num.intercepts(fit, type=c('fit', 'var', 'coef'))
```

**Arguments**

fit	a model fit object
type	the default is to return the formal number of intercepts used when fitting the model. Set type='var' to return the actual number of intercepts stored in the var object, or type='coef' to return the actual number in the fitted coefficients. The former will be less than the number fitted for <a href="#">orm</a> fits, and the latter for <a href="#">orm</a> fits passed through <a href="#">fit.mult.impute</a> . If the var object is not present, the number of intercepts is determined from the ab element of the info.matrix object if it is present.

**Value**

num.intercepts returns an integer with the number of intercepts in the model.

**See Also**

[orm](#), [fit.mult.impute](#)

---

ordGroupBoot	<i>Minimally Group an Ordinal Variable So Bootstrap Samples Will Contain All Distinct Values</i>
--------------	--

---

### Description

When bootstrapping models for ordinal  $Y$  when  $Y$  is fairly continuous, it is frequently the case that one or more bootstrap samples will not include one or more of the distinct original  $Y$  values. When fitting an ordinal model (including a Cox PH model), this means that an intercept cannot be estimated, and the parameter vectors will not align over bootstrap samples. To prevent this from happening, some grouping of  $Y$  may be necessary. The `ordGroupBoot` function uses `cutGn()` to group  $Y$  so that the minimum number in any group is guaranteed to not exceed a certain integer  $m$ . `ordGroupBoot` tries a range of  $m$  and stops at the lowest  $m$  such that either all  $B$  tested bootstrap samples contain all the original distinct values of  $Y$  (if  $B > 0$ ), or that the probability that a given sample of size  $n$  with replacement will contain all the distinct original values exceeds `aprob` ( $B = 0$ ). This probability is computed approximately using an approximation to the probability of complete sample coverage from the *coupon collector's problem* and is quite accurate for our purposes.

### Usage

```
ordGroupBoot(
  y,
  B = 0,
  m = 7:min(15, floor(n/3)),
  what = c("mean", "factor", "m"),
  aprob = 0.9999,
  pr = TRUE
)
```

### Arguments

<code>y</code>	a numeric vector
<code>B</code>	number of bootstrap samples to test, or zero to use a coverage probability approximation
<code>m</code>	range of minimum group sizes to test; the default range is usually adequate
<code>what</code>	specifies that either the mean $y$ in each group should be returned, a factor version of this with interval endpoints in the levels, or the computed value of $m$ should be returned
<code>aprob</code>	minimum coverage probability sought
<code>pr</code>	set to <code>FALSE</code> to not print the computed value of the minimum $m$ satisfying the needed condition

### Value

a numeric vector corresponding to  $y$  but grouped, containing either the mean of  $y$  in each group or a factor variable representing grouped  $y$ , either with the minimum  $m$  that satisfied the required sample coverage

**Author(s)**

Frank Harrell

**See Also**[cutGn\(\)](#)**Examples**

```
set.seed(1)
x <- c(1:6, NA, 7:22)
ordGroupBoot(x, m=5:10)
ordGroupBoot(x, m=5:10, B=5000, what='factor')
```

---

`pairUpDiff`*pairUpDiff*

---

**Description**

Pair-up and Compute Differences

**Usage**

```
pairUpDiff(
  x,
  major = NULL,
  minor = NULL,
  group,
  refgroup,
  lower = NULL,
  upper = NULL,
  minkeep = NULL,
  sortdiff = TRUE,
  conf.int = 0.95
)
```

**Arguments**

<code>x</code>	a numeric vector
<code>major</code>	an optional factor or character vector
<code>minor</code>	an optional factor or character vector
<code>group</code>	a required factor or character vector with two levels
<code>refgroup</code>	a character string specifying which level of group is to be subtracted
<code>lower</code>	an optional numeric vector giving the lower <code>conf.int</code> confidence limit for <code>x</code>
<code>upper</code>	similar to <code>lower</code> but for the upper limit

minkeep	the minimum value of $x$ required to keep the observation. An observation is kept if either group has $x$ exceeding or equalling minkeep. Default is to keep all observations.
sortdiff	set to FALSE to avoid sorting observations by descending between-group differences
conf.int	confidence level; must have been the value used to compute lower and upper if they are provided

### Details

This function sets up for plotting half-width confidence intervals for differences, sorting by descending order of differences within major categories, especially for dot charts as produced by `dotchartpl()`. Given a numeric vector  $x$  and a grouping (superpositioning) vector  $group$  with exactly two levels, computes differences in possibly transformed  $x$  between levels of group for the two observations that are equal on major and minor. If lower and upper are specified, using `conf.int` and approximate normality on the transformed scale to backsolve for the standard errors of estimates, and uses approximate normality to get confidence intervals on differences by taking the square root of the sum of squares of the two standard errors. Coordinates for plotting half-width confidence intervals are also computed. These intervals may be plotted on the same scale as  $x$ , having the property that they overlap the two  $x$  values if and only if there is no "significant" difference at the `conf.int` level.

### Value

a list of two objects both sorted by descending values of differences in  $x$ . The  $X$  object is a data frame that contains the original variables sorted by descending differences across group and in addition a variable `subscripts` denoting the subscripts of original observations with possible re-sorting and dropping depending on `sortdiff` and `minkeep`. The  $D$  data frame contains sorted differences (`diff`), `major`, `minor`, `sd` of difference, lower and upper confidence limits for the difference, `mid`, the midpoint of the two  $x$  values involved in the difference, `lowermid`, the midpoint minus 1/2 the width of the confidence interval, and `uppermid`, the midpoint plus 1/2 the width of the confidence interval. Another element returned is `dropped` which is a vector of major / minor combinations dropped due to `minkeep`.

### Author(s)

Frank Harrell

### Examples

```
x <- c(1, 4, 7, 2, 5, 3, 6)
pairUpDiff(x, c(rep('A', 4), rep('B', 3)),
  c('u','u','v','v','z','z','q'),
  c('a','b','a','b','a','b','a'), 'a', x-.1, x+.1)
```

panel.bpplot

*Box-Percentile Panel Function for Trellis***Description**

For all their good points, box plots have a high ink/information ratio in that they mainly display 3 quartiles. Many practitioners have found that the "outer values" are difficult to explain to non-statisticians and many feel that the notion of "outliers" is too dependent on (false) expectations that data distributions should be Gaussian.

panel.bpplot is a panel function for use with trellis, especially for bwplot. It draws box plots (without the whiskers) with any number of user-specified "corners" (corresponding to different quantiles), but it also draws box-percentile plots similar to those drawn by Jeffrey Banfield's (<umsfjban@bill.oscs.montana.edu>) bpplot function. To quote from Banfield, "box-percentile plots supply more information about the univariate distributions. At any height the width of the irregular 'box' is proportional to the percentile of that height, up to the 50th percentile, and above the 50th percentile the width is proportional to 100 minus the percentile. Thus, the width at any given height is proportional to the percent of observations that are more extreme in that direction. As in boxplots, the median, 25th and 75th percentiles are marked with line segments across the box."

panel.bpplot can also be used with base graphics to add extended box plots to an existing plot, by specifying nogrid=TRUE, height=...

panel.bpplot is a generalization of bpplot and panel.bwplot in that it works with trellis (making the plots horizontal so that category labels are more visible), it allows the user to specify the quantiles to connect and those for which to draw reference lines, and it displays means (by default using dots).

bpplt draws horizontal box-percentile plot much like those drawn by panel.bpplot but taking as the starting point a matrix containing quantiles summarizing the data. bpplt is primarily intended to be used internally by plot.summary.formula.reverse or plot.summaryM but when used with no arguments has a general purpose: to draw an annotated example box-percentile plot with the default quantiles used and with the mean drawn with a solid dot. This schematic plot is rendered nicely in postscript with an image height of 3.5 inches.

bppltp is like bpplt but for plotly graphics, and it does not draw an annotated extended box plot example.

bpplotM uses the lattice bwplot function to depict multiple numeric continuous variables with varying scales in a single lattice graph, after reshaping the dataset into a tall and thin format.

**Usage**

```
panel.bpplot(x, y, box.ratio=1, means=TRUE, qref=c(.5,.25,.75),
             probs=c(.05,.125,.25,.375), nout=0,
             nloc=c('right lower', 'right', 'left', 'none'), cex.n=.7,
             datadensity=FALSE, scat1d.opts=NULL,
             violin=FALSE, violin.opts=NULL,
             font=box.dot$font, pch=box.dot$pch,
             cex.means =box.dot$cex, col=box.dot$col,
```



```

        nogrid=NULL, height=NULL, ...)

# E.g. bwplot(formula, panel=panel.bplot, panel.bplot.parameters)

bpplt(stats, xlim, xlab='', box.ratio = 1, means=TRUE,
       qref=c(.5,.25,.75), qomit=c(.025,.975),
       pch=16, cex.labels=par('cex'), cex.points=if(prototype)1 else 0.5,
       grid=FALSE)

bpplt(p=plotly::plot_ly(),
      stats, xlim, xlab='', box.ratio = 1, means=TRUE,
      qref=c(.5,.25,.75), qomit=c(.025,.975),
      teststat=NULL, showlegend=TRUE)

bplotM(formula=NULL, groups=NULL, data=NULL, subset=NULL, na.action=NULL,
       qlim=0.01, xlim=NULL,
       nloc=c('right lower','right','left','none'),
       vnames=c('labels', 'names'), cex.n=.7, cex.strip=1,
       outerlabels=TRUE, ...)

```

### Arguments

x	continuous variable whose distribution is to be examined
y	grouping variable
box.ratio	see <a href="#">panel.bwplot</a>
means	set to FALSE to suppress drawing a character at the mean value
qref	vector of quantiles for which to draw reference lines. These do not need to be included in probs.
probs	vector of quantiles to display in the box plot. These should all be less than 0.5; the mirror-image quantiles are added automatically. By default, probs is set to c(.05, .125, .25, .375) so that intervals contain 0.9, 0.75, 0.5, and 0.25 of the data. To draw all 99 percentiles, i.e., to draw a box-percentile plot, set probs=seq(.01, .49, by=.01). To make a more traditional box plot, use probs=.25.
nout	tells the function to use scat1d to draw tick marks showing the nout smallest and nout largest values if nout >= 1, or to show all values less than the nout quantile or greater than the 1-nout quantile if 0 < nout <= 0.5. If nout is a whole number, only the first n/2 observations are shown on either side of the median, where n is the total number of observations.
nloc	location to plot number of non-NA observations next to each box. Specify nloc='none' to suppress. For panel.bplot, the default nloc is 'none' if nogrid=TRUE.
cex.n	character size for nloc
datadensity	set to TRUE to invoke scat1d to draw a data density (one-dimensional scatter diagram or rug plot) inside each box plot.
scat1d.opts	a list containing named arguments (without abbreviations) to pass to scat1d when datadensity=TRUE or nout > 0

violin	set to TRUE to invoke <code>panel.violin</code> in addition to drawing box-percentile plots
violin.opts	a list of options to pass to <code>panel.violin</code>
cex.means	character size for dots representing means
font, pch, col	see <a href="#">panel.bwplot</a>
nogrid	set to TRUE to use in base graphics
height	if <code>nogrid=TRUE</code> , specifies the height of the box in user y units
...	arguments passed to <code>points</code> or <code>panel.bplot</code> or <code>bwplot</code>
stats, xlim, xlab, qomit, cex.labels, cex.points, grid	undocumented arguments to <code>bpplt</code> . For <code>bpplotM</code> , <code>xlim</code> is a list with elements named as the x-axis variables, to override the <code>qlim</code> calculations with user-specified x-axis limits for selected variables. Example: <code>xlim=list(age=c(20,60))</code> .
p	an already-started <code>plotly</code> object
teststat	an html expression containing a test statistic
showlegend	set to TRUE to have <code>plotly</code> include a legend. Not recommended when plotting more than one variable.
formula	a formula with continuous numeric analysis variables on the left hand side and stratification variables on the right. The first variable on the right is the one that will vary the fastest, forming the y-axis. <code>formula</code> may be omitted, in which case all numeric variables with more than 5 unique values in data will be analyzed. Or <code>formula</code> may be a vector of variable names in data to analyze. In the latter two cases (and only those cases), groups must be given, representing a character vector with names of stratification variables.
groups	see above
data	an optional data frame
subset	an optional subsetting expression or logical vector
na.action	specifies a function to possibly subset the data according to NAs (default is no such subsetting).
qlim	the outer quantiles to use for scaling each panel in <code>bpplotM</code>
vnames	default is to use variable label attributes when they exist, or use variable names otherwise. Specify <code>vnames='names'</code> to always use variable names for panel labels in <code>bpplotM</code>
cex.strip	character size for panel strip labels
outerlabels	if TRUE, pass the <code>lattice</code> graphics through the <code>latticeExtra</code> package's <code>useOuterStrips</code> function if there are two conditioning (paneling) variables, to put panel labels in outer margins.

**Author(s)**

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University School of Medicine  
 <fh@fharrell.com>

**References**

Esty WW, Banfield J: The box-percentile plot. J Statistical Software 8 No. 17, 2003.

**See Also**

[bplot](#), [panel.bwplot](#), [scat1d](#), [quantile](#), [Ecdf](#), [summaryP](#), [useOuterStrips](#)

**Examples**

```
set.seed(13)
x <- rnorm(1000)
g <- sample(1:6, 1000, replace=TRUE)
x[g==1][1:20] <- rnorm(20)+3 # contaminate 20 x's for group 1

# default trellis box plot
require(lattice)
bwplot(g ~ x)

# box-percentile plot with data density (rug plot)
bwplot(g ~ x, panel=panel.bplot, probs=seq(.01,.49,by=.01), datadensity=TRUE)
# add ,scat1d.opts=list(tfrac=1) to make all tick marks the same size
# when a group has > 125 observations

# small dot for means, show only .05,.125,.25,.375,.625,.75,.875,.95 quantiles
bwplot(g ~ x, panel=panel.bplot, cex.means=.3)

# suppress means and reference lines for lower and upper quartiles
bwplot(g ~ x, panel=panel.bplot, probs=c(.025,.1,.25), means=FALSE, qref=FALSE)

# continuous plot up until quartiles ("Tootsie Roll plot")
bwplot(g ~ x, panel=panel.bplot, probs=seq(.01,.25,by=.01))

# start at quartiles then make it continuous ("coffin plot")
bwplot(g ~ x, panel=panel.bplot, probs=seq(.25,.49,by=.01))

# same as previous but add a spike to give 0.95 interval
bwplot(g ~ x, panel=panel.bplot, probs=c(.025,seq(.25,.49,by=.01)))

# decile plot with reference lines at outer quintiles and median
bwplot(g ~ x, panel=panel.bplot, probs=c(.1,.2,.3,.4), qref=c(.5,.2,.8))

# default plot with tick marks showing all observations outside the outer
# box (.05 and .95 quantiles), with very small ticks
```

```

bwplot(g ~ x, panel=panel.bplot, nout=.05, scat1d.opts=list(frac=.01))

# show 5 smallest and 5 largest observations
bwplot(g ~ x, panel=panel.bplot, nout=5)

# Use a scat1d option (preserve=TRUE) to ensure that the right peak extends
# to the same position as the extreme scat1d
bwplot(~x , panel=panel.bplot, probs=seq(.00,.5,by=.001),
       datadensity=TRUE, scat1d.opt=list(preserve=TRUE))

# Add an extended box plot to an existing base graphics plot
plot(x, 1:length(x))
panel.bplot(x, 1070, nogrid=TRUE, pch=19, height=15, cex.means=.5)

# Draw a prototype showing how to interpret the plots
bplot()

# Example for bplotM
set.seed(1)
n <- 800
d <- data.frame(treatment=sample(c('a','b'), n, TRUE),
               sex=sample(c('female','male'), n, TRUE),
               age=rnorm(n, 40, 10),
               bp =rnorm(n, 120, 12),
               wt =rnorm(n, 190, 30))
label(d$bp) <- 'Systolic Blood Pressure'
units(d$bp) <- 'mmHg'
bplotM(age + bp + wt ~ treatment, data=d)
bplotM(age + bp + wt ~ treatment * sex, data=d, cex.strip=.8)
bplotM(age + bp + wt ~ treatment*sex, data=d,
       violin=TRUE,
       violin.opts=list(col=adjustcolor('blue', alpha.f=.15),
                       border=FALSE))

bplotM(c('age', 'bp', 'wt'), groups='treatment', data=d)
# Can use Hmisc Cs function, e.g. Cs(age, bp, wt)
bplotM(age + bp + wt ~ treatment, data=d, nloc='left')

# Without treatment: bplotM(age + bp + wt ~ 1, data=d)

## Not run:
# Automatically find all variables that appear to be continuous
getHdata(support)
bplotM(data=support, group='dzgroup',
       cex.strip=.4, cex.means=.3, cex.n=.45)

# Separate displays for categorical vs. continuous baseline variables
getHdata(pbc)
pbc <- upData(pbc, moveUnits=TRUE)

```

```

s <- summaryM(stage + sex + spiders ~ drug, data=pbcc)
plot(s)
Key(0, .5)
s <- summaryP(stage + sex + spiders ~ drug, data=pbcc)
plot(s, val ~ freq | var, groups='drug', pch=1:3, col=1:3,
      key=list(x=.6, y=.8))

bplotM(bili + albumin + protime + age ~ drug, data=pbcc)

## End(Not run)

```

---

partition

*Partitions an object into different sets*


---

### Description

Partitions an object into subsets of length defined in the sep argument.

### Usage

```

partition.vector(x, sep, ...)
partition.matrix(x, rowsep, colsep, ...)

```

### Arguments

x	object to be partitioned.
sep	determines how many elements should go into each set. The sum of sep should be equal to the length of x.
rowsep	determines how many rows should go into each set. The sum of rowsep must equal the number of rows in x.
colsep	determines how many columns should go into each set. The sum of colsep must equal the number of columns in x.
...	arguments used in other methods of partition.

### Value

A list of equal length as sep containing the partitioned objects.

### Author(s)

Charles Dupont

### See Also

[split](#)

**Examples**

```
a <- 1:7
partition.vector(a, sep=c(1,3,2,1))
```

---

pc1

*First Principal Component*

---

**Description**

Given a numeric matrix which may or may not contain NAs, `pc1` standardizes the columns to have mean 0 and variance 1 and computes the first principal component using `prcomp`. The proportion of variance explained by this component is printed, and so are the coefficients of the original (not scaled) variables. These coefficients may be applied to the raw data to obtain the first PC.

**Usage**

```
pc1(x, hi)
```

**Arguments**

<code>x</code>	numeric matrix
<code>hi</code>	if specified, the first PC is scaled so that its maximum value is <code>hi</code> and its minimum value is zero

**Value**

The vector of observations with the first PC. An attribute "coef" is attached to this vector. "coef" contains the raw-variable coefficients.

**Author(s)**

Frank Harrell

**See Also**

[prcomp](#)

**Examples**

```
set.seed(1)
x1 <- rnorm(100)
x2 <- x1 + rnorm(100)
w <- pc1(cbind(x1,x2))
attr(w, 'coef')
```

---

plot.princmp	<i>plot.princmp</i>
--------------	---------------------

---

**Description**

Plot Method for princmp

**Usage**

```
## S3 method for class 'princmp'
plot(
  x,
  which = c("scree", "loadings"),
  k = x$k,
  offset = 0.8,
  col = 1,
  adj = 0,
  ylim = NULL,
  add = FALSE,
  abbrev = 25,
  nrow = NULL,
  ...
)
```

**Arguments**

x	results of 'princmp'
which	'scree' or 'loadings'
k	number of components to show, default is 'k' specified to 'princmp'
offset	controls positioning of text labels for cumulative fraction of variance explained
col	color of plotted text in scree plot
adj	angle for plotting text in scree plot
ylim	y-axis scree plotting limits, a 2-vector
add	set to 'TRUE' to add a line to an existing scree plot without drawing axes
abbrev	an integer specifying the variable name length above which names are passed through [abbreviate(..., minlength=abbrev)]
nrow	number of rows to use in plotting loadings. Defaults to the 'ggplot2' 'facet_wrap' default.
...	unused

**Details**

Uses base graphics to by default plot the scree plot from a [princmp()] result, showing cumulative proportion of variance explained. Alternatively the standardized PC loadings are shown in a 'ggplot2' bar chart.

**Value**

'ggplot2' object if 'which='loadings'

**Author(s)**

Frank Harrell

---

plotCorrM

*plotCorrM*

---

**Description**

Plot Correlation Matrix and Correlation vs. Time Gap

**Usage**

```
plotCorrM(
  r,
  what = c("plots", "data"),
  type = c("rectangle", "circle"),
  xlab = "",
  ylab = "",
  maxsize = 12,
  xangle = 0
)
```

**Arguments**

r	correlation matrix
what	specifies whether to return plots or the data frame used in making the plots
type	specifies whether to use bottom-aligned rectangles (the default) or centered circles
xlab	x-axis label for correlation matrix
ylab	y-axis label for correlation matrix
maxsize	maximum circle size if type='circle'
xangle	angle for placing x-axis labels, defaulting to 0. Consider using xangle=45 when labels are long.

**Details**

Constructs two ggplot2 graphics. The first is a half matrix of rectangles where the height of the rectangle is proportional to the absolute value of the correlation coefficient, with positive and negative coefficients shown in different colors. The second graphic is a variogram-like graph of correlation coefficients on the y-axis and absolute time gap on the x-axis, with a loess smoother added. The times are obtained from the correlation matrix's row and column names if these are numeric. If any names are not numeric, the times are taken as the integers 1, 2, 3, ... The two graphics are ggplotly-ready if you use plotly::ggplotly(..., tooltip='label').



**Value**

a list containing two ggplot2 objects if what='plots', or a data frame if what='data'

**Author(s)**

Frank Harrell

**Examples**

```
set.seed(1)
r <- cor(matrix(rnorm(100), ncol=10))
g <- plotCorrM(r)
g[[1]] # plot matrix
g[[2]] # plot correlation vs gap time
# ggplotlyr(g[[2]])
# ggplotlyr uses ggplotly with tooltip='label' then removes
# txt: from hover text
```

---

plotCorrPrecision      *Plot Precision of Estimate of Pearson Correlation Coefficient*

---

**Description**

This function plots the precision (margin of error) of the product-moment linear correlation coefficient  $r$  vs. sample size, for a given vector of correlation coefficients  $\rho$ . Precision is defined as the larger of the upper confidence limit minus  $\rho$  and  $\rho$  minus the lower confidence limit. `labcurve` is used to automatically label the curves.

**Usage**

```
plotCorrPrecision(rho = c(0, 0.5), n = seq(10, 400, length.out = 100),
                 conf.int = 0.95, offset=0.025, ...)
```

**Arguments**

<code>rho</code>	single or vector of true correlations. A worst-case precision graph results from <code>rho=0</code>
<code>n</code>	vector of sample sizes to use on the x-axis
<code>conf.int</code>	confidence coefficient; default uses 0.95 confidence limits
<code>offset</code>	see <a href="#">labcurve</a>
<code>...</code>	other arguments to <a href="#">labcurve</a>

**Author(s)**

Xing Wang and Frank Harrell

**See Also**[rcorr, cor, cor.test](#)**Examples**

```
plotCorrPrecision()  
plotCorrPrecision(rho=0)
```

---

`plotlyM`*plotly Multiple*

---

**Description**

Generates multiple plotly graphics, driven by specs in a data frame

**Usage**

```
plotlyM(  
  data,  
  x = ~x,  
  y = ~y,  
  xhi = ~xhi,  
  yhi = ~yhi,  
  htext = NULL,  
  multplot = NULL,  
  strata = NULL,  
  fitter = NULL,  
  color = NULL,  
  size = NULL,  
  showpts = !length(fitter),  
  rotate = FALSE,  
  xlab = NULL,  
  ylab = NULL,  
  ylabpos = c("top", "y"),  
  xlim = NULL,  
  ylim = NULL,  
  shareX = TRUE,  
  shareY = FALSE,  
  height = NULL,  
  width = NULL,  
  nrows = NULL,  
  ncols = NULL,  
  colors = NULL,  
  alphaSegments = 1,  
  alphaCline = 0.3,  
  digits = 4,  
  zeroline = TRUE  
)
```

**Arguments**

<code>data</code>	input data frame
<code>x</code>	formula specifying the x-axis variable
<code>y</code>	formula for y-axis variable
<code>xhi</code>	formula for upper x variable limits (x taken to be lower value)
<code>yhi</code>	formula for upper y variable limit (y taken to be lower value)
<code>htext</code>	formula for hovertext variable
<code>multplot</code>	formula specifying a variable in <code>data</code> that when stratified on produces a separate plot
<code>strata</code>	formula specifying an optional stratification variable
<code>fitter</code>	a fitting such as <code>loess</code> that comes with a <code>predict</code> method. Alternatively specify <code>fitter='ecdf'</code> to use an internal function for computing and displaying ECDFs, which moves the analysis variable from the y-axis to the x-axis
<code>color</code>	plotly formula specifying a color variable or e.g. <code>~ I('black')</code> . To keep colors constant over multiple plots you will need to specify an <code>AsIs</code> color when you don't have a variable representing color groups.
<code>size</code>	plotly formula specifying a symbol size variable or <code>AsIs</code>
<code>showpts</code>	if <code>fitter</code> is given, set to <code>TRUE</code> to show raw data points in addition to smooth fits
<code>rotate</code>	set to <code>TRUE</code> to reverse the roles of x and y, for example to get horizontal dot charts with error bars
<code>xlab</code>	x-axis label. May contain html.
<code>ylab</code>	a named vector of y-axis labels, possibly containing html (see example below). The names of the vector must correspond to levels of the <code>multplot</code> variable. <code>ylab</code> can be unnamed if <code>multplot</code> is not used.
<code>ylabpos</code>	position of y-axis labels. Default is on top left of plot. Specify <code>ylabpos='y'</code> for usual y-axis placement.
<code>xlim</code>	2-vector of x-axis limits, optional
<code>ylim</code>	2-vector of y-axis limits, optional
<code>shareX</code>	specifies whether x-axes should be shared when they align vertically over multiple plots
<code>shareY</code>	specifies whether y-axes should be shared when they align horizontally over multiple plots
<code>height</code>	height of the combined image in pixels
<code>width</code>	width of the combined image in pixels
<code>nrows</code>	the number of rows to produce using <code>subplot</code>
<code>ncols</code>	the number of columns to produce using <code>subplot</code> (specify at most one of <code>nrows</code> , <code>ncols</code> )
<code>colors</code>	the color palette. Leave unspecified to use the default plotly palette
<code>alphaSegments</code>	alpha transparency for line segments (when <code>xhi</code> or <code>yhi</code> is not <code>NA</code> )
<code>alphaCline</code>	alpha transparency for lines used to connect points
<code>digits</code>	number of significant digits to use in constructing hovertext
<code>zeroline</code>	set to <code>FALSE</code> to suppress vertical line at <code>x=0</code>

## Details

Generates multiple plotly traces and combines them with `plotly::subplot`. The traces are controlled by specifications in data frame `data` plus various arguments. `data` must contain these variables: `x`, `y`, and `tracename` (if `color` is not an "AsIs" color such as `~ I('black')`), and can contain these optional variables: `xhi`, `yhi` (rows containing NA for both `xhi` and `yhi` represent points, and those with non-NA `xhi` or `yhi` represent segments), `connect` (set to `TRUE` for rows for points, to connect the symbols), `legendgroup` (see plotly documentation), and `htext` (hovertext). If the `color` argument is given and it is not an "AsIs" color, the variable named in the `color` formula must also be in `data`. Likewise for `size`. If the `multplot` is given, the variable given in the formula must be in `data`. If `strata` is present, another level of separate plots is generated by levels of `strata`, within levels of `multplot`.

If `fitter` is specified, `x,y` coordinates for an individual plot are run through `fitter`, and a line plot is made instead of showing data points. Alternatively you can specify `fitter='ecdf'` to compute and plot emirical cumulative distribution functions.

## Value

plotly object produced by `subplot`

## Author(s)

Frank Harrell

## Examples

```
## Not run:
set.seed(1)
pts <- expand.grid(v=c('y1', 'y2', 'y3'), x=1:4, g=c('a', 'b'), yhi=NA,
                 tracename='mean', legendgroup='mean',
                 connect=TRUE, size=4)

pts$y <- round(runif(nrow(pts)), 2)

segs <- expand.grid(v=c('y1', 'y2', 'y3'), x=1:4, g=c('a', 'b'),
                 tracename='limits', legendgroup='limits',
                 connect=NA, size=6)

segs$y <- runif(nrow(pts))
segs$yhi <- segs$y + runif(nrow(pts), .05, .15)

z <- rbind(pts, segs)

xlab <- labelPlotmath('X<sub>12</sub>', 'm/sec<sup>2</sup>', html=TRUE)
ylab <- c(y1=labelPlotmath('Y1', 'cm', html=TRUE),
        y2='Y2',
        y3=labelPlotmath('Y3', 'mm', html=TRUE))

W=plotlyM(z, multplot=~v, color=~g, xlab=xlab, ylab=ylab, ncols=2,
         colors=c('black', 'blue'))

W2=plotlyM(z, multplot=~v, color=~I('black'), xlab=xlab, ylab=ylab,
```

```
colors=c('black', 'blue'))
```

```
## End(Not run)
```

---

plsmo

*Plot smoothed estimates*

---

## Description

Plot smoothed estimates of  $x$  vs.  $y$ , handling missing data for `lowess` or `supsmu`, and adding axis labels. Optionally suppresses plotting extrapolated estimates. An optional group variable can be specified to compute and plot the smooth curves by levels of group. When group is present, the `datadensity` option will draw tick marks showing the location of the raw  $x$ -values, separately for each curve. `plsmo` has an option to plot connected points for raw data, with no smoothing. The non-panel version of `plsmo` allows  $y$  to be a matrix, for which smoothing is done separately over its columns. If both group and multi-column  $y$  are used, the number of curves plotted is the product of the number of groups and the number of  $y$  columns.

`method='intervals'` is often used when  $y$  is binary, as it may be tricky to specify a reasonable smoothing parameter to `lowess` or `supsmu` in this case. The 'intervals' method uses the `cutGn` function to form intervals of  $x$  containing a minimum of `mobs` observations. For each interval the `ifun` function summarizes  $y$ , with the default being the mean (proportions for binary  $y$ ). The results are plotted as step functions, with vertical discontinuities drawn with a saturation of 0.15 of the original color. A plus sign is drawn at the mean  $x$  within each interval. For this approach, the default  $x$ -range is the entire raw data range, and `trim` and `evaluate` are ignored. For `panel.plsmo` it is best to specify `type='l'` when using 'intervals'.

`panel.plsmo` is a panel function for `trellis` for the `xyplot` function that uses `plsmo` and its options to draw one or more nonparametric function estimates on each panel. This has advantages over using `xyplot` with `panel.xyplot` and `panel.loess`: (1) by default it will invoke `labcurve` to label the curves where they are most separated, (2) the `datadensity` option will put rug plots on each curve (instead of a single rug plot at the bottom of the graph), and (3) when `panel.plsmo` invokes `plsmo` it can use the "super smoother" (`supsmu` function) instead of `lowess`, or pass `method='intervals'`. `panel.plsmo` senses when a group variable is specified to `xyplot` so that it can invoke `panel.superpose` instead of `panel.xyplot`. Using `panel.plsmo` through `trellis` has some advantages over calling `plsmo` directly in that conditioning variables are allowed and `trellis` uses nicer fonts etc.

When a group variable was used, `panel.plsmo` creates a function `Key` in the session frame that the user can invoke to draw a key for individual data point symbols used for the groups. By default, the key is positioned at the upper right corner of the graph. If `Key(Locator(1))` is specified, the key will appear so that its upper left corner is at the coordinates of the mouse click.

For `ggplot2` graphics the counterparts are `stat_plsmo` and `histSpikeg`.

## Usage

```
plsmo(x, y, method=c("lowess", "supsmu", "raw", "intervals"), xlab, ylab,
      add=FALSE, lty=1 : lc, col=par("col"), lwd=par("lwd"),
```

```

iter=if(length(unique(y))>2) 3 else 0, bass=0, f=2/3, mobs=30, trim,
fun, ifun=mean, group, prefix, xlim, ylim,
label.curves=TRUE, datadensity=FALSE, scat1d.opts=NULL,
lines.=TRUE, subset=TRUE,
grid=FALSE, evaluate=NULL, ...)

```

#To use panel function:

```

#xyplot(formula=y ~ x | conditioningvars, groups,
#       panel=panel.plsmo, type='b',
#       label.curves=TRUE,
#       lwd = superpose.line$lwd,
#       lty = superpose.line$lty,
#       pch = superpose.symbol$pch,
#       cex = superpose.symbol$cex,
#       font = superpose.symbol$font,
#       col = NULL, scat1d.opts=NULL, \dots)

```

### Arguments

x	vector of x-values, NAs allowed
y	vector or matrix of y-values, NAs allowed
method	"lowess" (the default), "supsmu", "raw" to not smooth at all, or "intervals" to use intervals (see above)
xlab	x-axis label iff add=F. Defaults of label(x) or argument name.
ylab	y-axis label, like xlab.
add	Set to T to call lines instead of plot. Assumes axes already labeled.
lty	line type, default=1,2,3,..., corresponding to columns of y and group combinations
col	color for each curve, corresponding to group. Default is current par("col").
lwd	vector of line widths for the curves, corresponding to group. Default is current par("lwd"). lwd can also be specified as an element of label.curves if label.curves is a list.
iter	iter parameter if method="lowess", default=0 if y is binary, and 3 otherwise.
bass	bass parameter if method="supsmu", default=0.
f	passed to the lowess function, for method="lowess"
mobs	for method='intervals', the minimum number of observations per interval
trim	only plots smoothed estimates between trim and 1-trim quantiles of x. Default is to use 10th smallest to 10th largest x in the group if the number of observations in the group exceeds 200 (0 otherwise). Specify trim=0 to plot over entire range.
fun	after computing the smoothed estimates, if fun is given the y-values are transformed by fun()
ifun	a summary statistic function to apply to the y-variable for method='intervals'. Default is mean.

group	a variable, either a factor vector or one that will be converted to factor by <code>plsmo</code> , that is used to stratify the data so that separate smooths may be computed
prefix	a character string to appear in group of group labels. The presence of <code>prefix</code> ensures that <code>labcurve</code> will be called even when <code>add=TRUE</code> .
xlim	a vector of 2 x-axis limits. Default is observed range.
ylim	a vector of 2 y-axis limits. Default is observed range.
label.curves	set to <code>FALSE</code> to prevent <code>labcurve</code> from being called to label multiple curves corresponding to groups. Set to a list to pass options to <code>labcurve</code> . <code>lty</code> and <code>col</code> are passed to <code>labcurve</code> automatically.
datadensity	set to <code>TRUE</code> to draw tick marks on each curve, using x-coordinates of the raw data x values. This is done using <code>scat1d</code> .
scat1d.opts	a list of options to hand to <code>scat1d</code>
lines.	set to <code>FALSE</code> to suppress smoothed curves from being drawn. This can make sense if <code>datadensity=TRUE</code> .
subset	a logical or integer vector specifying a subset to use for processing, with respect too all variables being analyzed
grid	set to <code>TRUE</code> if the <code>R grid</code> package drew the current plot
evaluate	number of points to keep from smoother. If specified, an equally-spaced grid of <code>evaluate</code> x values will be obtained from the smoother using linear interpolation. This will keep from plotting an enormous number of points if the dataset contains a very large number of unique x values.
...	optional arguments that are passed to <code>scat1d</code> , or optional parameters to pass to <code>plsmo</code> from <code>panel.plsmo</code> . See optional arguments for <code>plsmo</code> above.
type	set to <code>p</code> to have <code>panel.plsmo</code> plot points (and not call <code>plsmo</code> ), <code>l</code> to call <code>plsmo</code> and not plot points, or use the default <code>b</code> to plot both.
pch, cex, font	vectors of graphical parameters corresponding to the groups (scalars if group is absent). By default, the parameters set up by <code>trellis</code> will be used.

### Value

`plsmo` returns a list of curves (x and y coordinates) that was passed to `labcurve`

### Side Effects

plots, and `panel.plsmo` creates the `Key` function in the session frame.

### See Also

[lowess](#), [supsmu](#), [label](#), [quantile](#), [labcurve](#), [scat1d](#), [xyplot](#), [panel.superpose](#), [panel.xyplot](#), [stat\\_plsmo](#), [histSpikeg](#), [cutGn](#)

**Examples**

```

set.seed(1)
x <- 1:100
y <- x + runif(100, -10, 10)
plsmo(x, y, "supsmu", xlab="Time of Entry")
#Use label(y) or "y" for ylab

plsmo(x, y, add=TRUE, lty=2)
#Add lowess smooth to existing plot, with different line type

age <- rnorm(500, 50, 15)
survival.time <- rexp(500)
sex <- sample(c('female','male'), 500, TRUE)
race <- sample(c('black','non-black'), 500, TRUE)
plsmo(age, survival.time < 1, fun=qlogis, group=sex) # plot logit by sex

#Bivariate Y
sbp <- 120 + (age - 50)/10 + rnorm(500, 0, 8) + 5 * (sex == 'male')
dbp <- 80 + (age - 50)/10 + rnorm(500, 0, 8) - 5 * (sex == 'male')
Y <- cbind(sbp, dbp)
plsmo(age, Y)
plsmo(age, Y, group=sex)

#Plot points and smooth trend line using trellis
# (add type='l' to suppress points or type='p' to suppress trend lines)
require(lattice)
xyplot(survival.time ~ age, panel=panel.plsmo)

#Do this for multiple panels
xyplot(survival.time ~ age | sex, panel=panel.plsmo)

#Repeat this using equal sample size intervals (n=25 each) summarized by
#the median, then a proportion (mean of binary y)
xyplot(survival.time ~ age | sex, panel=panel.plsmo, type='l',
       method='intervals', mobs=25, ifun=median)
ybinary <- ifelse(runif(length(sex)) < 0.5, 1, 0)
xyplot(ybinary ~ age, groups=sex, panel=panel.plsmo, type='l',
       method='intervals', mobs=75, ifun=mean, xlim=c(0, 120))

#Do this for subgroups of points on each panel, show the data
#density on each curve, and draw a key at the default location
xyplot(survival.time ~ age | sex, groups=race, panel=panel.plsmo,
       datadensity=TRUE)
Key()

#Use wloess.noiter to do a fast weighted smooth

```



```

plot(x, y)
lines(wtd.loess.noiter(x, y))
lines(wtd.loess.noiter(x, y, weights=c(rep(1,50), 100, rep(1,49))), col=2)
points(51, y[51], pch=18) # show overly weighted point
#Try to duplicate this smooth by replicating 51st observation 100 times
lines(wtd.loess.noiter(c(x,rep(x[51],99)),c(y,rep(y[51],99)),
  type='ordered all'), col=3)
#Note: These two don't agree exactly

```

pMedian

*pMedian***Description**

Pseudomedian

**Usage**

```
pMedian(x, na.rm = FALSE)
```

**Arguments**

x	a numeric vector
na.rm	set to TRUE to exclude NAs before computing the pseudomedian

**Details**

Uses fast Fortran code to compute the pseudomedian of a numeric vector. The pseudomedian is the median of all possible midpoints of two observations. The pseudomedian is also called the Hodges-Lehmann one-sample estimator. The Fortran code is was originally from JF Monahan, and was converted to C++ in the DescTools package. It has been converted to Fortran 2018 here.

**Value**

a scalar numeric value

**See Also**

<https://dl.acm.org/doi/10.1145/1271.319414>, <https://www4.stat.ncsu.edu/~monahan/jul10/>

**Examples**

```

x <- c(1:4, 10000)
pMedian(x)
# Compare with brute force calculation and with wilcox.test
w <- outer(x, x, '+')
median(w[lower.tri(w, diag=TRUE)]) / 2
wilcox.test(x, conf.int=TRUE)

```

## Description

popower computes the power for a two-tailed two sample comparison of ordinal outcomes under the proportional odds ordinal logistic model. The power is the same as that of the Wilcoxon test but with ties handled properly. posamsize computes the total sample size needed to achieve a given power. Both functions compute the efficiency of the design compared with a design in which the response variable is continuous. print methods exist for both functions. Any of the input arguments may be vectors, in which case a vector of powers or sample sizes is returned. These functions use the methods of Whitehead (1993).

pomodm is a function that assists in translating odds ratios to differences in mean or median on the original scale.

simPOcuts simulates simple unadjusted two-group comparisons under a PO model to demonstrate the natural sampling variability that causes estimated odds ratios to vary over cutoffs of Y.

propsPO uses ggplot2 to plot a stacked bar chart of proportions stratified by a grouping variable (and optionally a stratification variable), with an optional additional graph showing what the proportions would be had proportional odds held and an odds ratio was applied to the proportions in a reference group. If the result is passed to ggplotly, customized tooltip hover text will appear.

propsTrans uses ggplot2 to plot all successive transition proportions. formula has the state variable on the left hand side, the first right-hand variable is time, and the second right-hand variable is a subject ID variable.

multEventChart uses ggplot2 to plot event charts showing state transitions, account for absorbing states/events. It is based on code written by Lucy D'Agostino McGowan posted at <https://livefreeordichotomize.com/posts/2020-05-21-survival-model-detective-1/>.

## Usage

```
popower(p, odds.ratio, n, n1, n2, alpha=0.05)
## S3 method for class 'popower'
print(x, ...)
posamsize(p, odds.ratio, fraction=.5, alpha=0.05, power=0.8)
## S3 method for class 'posamsize'
print(x, ...)
pomodm(x=NULL, p, odds.ratio=1)
simPOcuts(n, nsim=10, odds.ratio=1, p)
propsPO(formula, odds.ratio=NULL, ref=NULL, data=NULL, ncol=NULL, nrow=NULL )
propsTrans(formula, data=NULL, labels=NULL, arrow='\u2794',
            maxsize=12, ncol=NULL, nrow=NULL)
multEventChart(formula, data=NULL, absorb=NULL, sortbylast=FALSE,
               colorTitle=label(y), eventTitle='Event',
               palette='OrRd',
               eventSymbols=c(15, 5, 1:4, 6:10),
               timeInc=min(diff(unique(x))/2))
```

**Arguments**

<code>p</code>	a vector of marginal cell probabilities which must add up to one. For <code>popower</code> and <code>posamsize</code> , The <i>i</i> th element specifies the probability that a patient will be in response level <i>i</i> , averaged over the two treatment groups. For <code>pomodm</code> and <code>simPOcuts</code> , <code>p</code> is the vector of cell probabilities to be translated under a given odds ratio. For <code>simPOcuts</code> , if <code>p</code> has names, those names are taken as the ordered distinct Y-values. Otherwise Y-values are taken as the integers 1, 2, ... up to the length of <code>p</code> .
<code>odds.ratio</code>	the odds ratio to be able to detect. It doesn't matter which group is in the numerator. For <code>propsPO</code> , <code>odds.ratio</code> is a function of the grouping (right hand side) variable value. The value of the function specifies the odds ratio to apply to the reference group to get all other group's expected proportions were proportional odds to hold against the first group. Normally the function should return 1.0 when its <code>x</code> argument corresponds to the <code>ref</code> group. For <code>pomodm</code> and <code>simPOcuts</code> is the odds ratio to apply to convert the given cell probabilities.
<code>n</code>	total sample size for <code>popower</code> . You must specify either <code>n</code> or <code>n1</code> and <code>n2</code> . If you specify <code>n</code> , <code>n1</code> and <code>n2</code> are set to <code>n/2</code> . For <code>simPOcuts</code> is a single number equal to the combined sample sizes of two groups.
<code>n1</code>	for <code>popower</code> , the number of subjects in treatment group 1
<code>n2</code>	for <code>popower</code> , the number of subjects in group 2
<code>nsim</code>	number of simulated studies to create by <code>simPOcuts</code>
<code>alpha</code>	type I error
<code>x</code>	an object created by <code>popower</code> or <code>posamsize</code> , or a vector of data values given to <code>pomodm</code> that corresponds to the vector <code>p</code> of probabilities. If <code>x</code> is omitted for <code>pomodm</code> , the <code>odds.ratio</code> will be applied and the new vector of individual probabilities will be returned. Otherwise if <code>x</code> is given to <code>pomodm</code> , a 2-vector with the mean and median <code>x</code> after applying the odds ratio is returned.
<code>fraction</code>	for <code>posamsize</code> , the fraction of subjects that will be allocated to group 1
<code>power</code>	for <code>posamsize</code> , the desired power (default is 0.8)
<code>formula</code>	an R formula expression for <code>propsPO</code> where the outcome categorical variable is on the left hand side and the grouping variable is on the right. It is assumed that the left hand variable is either already a factor or will have its levels in the right order for an ordinal model when it is converted to factor. For <code>multEventChart</code> the left hand variable is a categorical status variable, the first right hand side variable represents time, and the second right side variable is a unique subject ID. One line is produced per subject.
<code>ref</code>	for <code>propsPO</code> specifies the reference group (value of the right hand side formula variable) to use in computing proportions on which to translate proportions in other groups, under the proportional odds assumption.
<code>data</code>	a data frame or <code>data.table</code>
<code>labels</code>	for <code>propsTrans</code> is an optional character vector corresponding to <code>y=1,2,3,...</code> that is used to construct <code>plotly</code> <code>hovertext</code> as a <code>label</code> attribute in the <code>ggplot2</code> aesthetic. Used with <code>y</code> is integer on axes but you want long labels in <code>hovertext</code> .

arrow	character to use as the arrow symbol for transitions in propsTrans. The default is the dingbats heavy wide-headed rightwards arrow.
nrow, ncol	see <a href="#">facet_wrap</a>
maxsize	maximum symbol size
...	unused
absorb	character vector specifying the subset of levels of the left hand side variable that are absorbing states such as death or hospital discharge
sortbylast	set to TRUE to sort the subjects by the severity of the status at the last time point
colorTitle	label for legend for status
eventTitle	label for legend for absorb
palette	a single character string specifying the <a href="#">scale_fill_brewer</a> color palette
eventSymbols	vector of symbol codes. Default for first two symbols is a solid square and an open diamond.
timeInc	time increment for the x-axis. Default is 1/2 the shortest gap between any two distinct times in the data.

**Value**

a list containing power, eff (relative efficiency), and approx.se (approximate standard error of log odds ratio) for popower, or containing n and eff for posamsize.

**Author(s)**

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University School of Medicine  
 <fh@fharrell.com>

**References**

Whitehead J (1993): Sample size calculations for ordered categorical data. *Stat in Med* 12:2257–2271.

Julious SA, Campbell MJ (1996): Letter to the Editor. *Stat in Med* 15: 1065–1066. Shows accuracy of formula for binary response case.

**See Also**

[simRegOrd](#), [bpower](#), [cpower](#), [impactPO](#)

**Examples**

```
# For a study of back pain (none, mild, moderate, severe) here are the
# expected proportions (averaged over 2 treatments) that will be in
# each of the 4 categories:
```

```
p <- c(.1, .2, .4, .3)
```

```

popower(p, 1.2, 1000) # OR=1.2, total n=1000
posamsize(p, 1.2)
popower(p, 1.2, 3148)
# If p was the vector of probabilities for group 1, here's how to
# compute the average over the two groups:
# p2 <- pomodm(p=p, odds.ratio=1.2)
# pavg <- (p + p2) / 2

# Compare power to test for proportions for binary case,
# proportion of events in control group of 0.1
p <- 0.1; or <- 0.85; n <- 4000
popower(c(1 - p, p), or, n) # 0.338
bpower(p, odds.ratio=or, n=n) # 0.320
# Add more categories, starting with 0.1 in middle
p <- c(.8, .1, .1)
popower(p, or, n) # 0.543
p <- c(.7, .1, .1, .1)
popower(p, or, n) # 0.67
# Continuous scale with final level have prob. 0.1
p <- c(rep(1 / n, 0.9 * n), 0.1)
popower(p, or, n) # 0.843

# Compute the mean and median x after shifting the probability
# distribution by an odds ratio under the proportional odds model
x <- 1 : 5
p <- c(.05, .2, .2, .3, .25)
# For comparison make up a sample that looks like this
X <- rep(1 : 5, 20 * p)
c(mean=mean(X), median=median(X))
pomodm(x, p, odds.ratio=1) # still have to figure out the right median
pomodm(x, p, odds.ratio=0.5)

# Show variation of odds ratios over possible cutoffs of Y even when P0
# truly holds. Run 5 simulations for a total sample size of 300.
# The two groups have 150 subjects each.
s <- simPOcuts(300, nsim=5, odds.ratio=2, p=p)
round(s, 2)

# An ordinal outcome with levels a, b, c, d, e is measured at 3 times
# Show the proportion of values in each outcome category stratified by
# time. Then compute what the proportions would be had the proportions
# at times 2 and 3 been the proportions at time 1 modified by two odds ratios

set.seed(1)
d <- expand.grid(time=1:3, reps=1:30)
d$y <- sample(letters[1:5], nrow(d), replace=TRUE)
propsP0(y ~ time, data=d, odds.ratio=function(time) c(1, 2, 4)[time])
# To show with plotly, save previous result as object p and then:
# plotly::ggplotly(p, tooltip='label')

# Add a stratification variable and don't consider an odds ratio
d <- expand.grid(time=1:5, sex=c('female', 'male'), reps=1:30)
d$y <- sample(letters[1:5], nrow(d), replace=TRUE)

```

```

propsP0(y ~ time + sex, data=d) # may add nrow= or ncol=

# Show all successive transition proportion matrices
d <- expand.grid(id=1:30, time=1:10)
d$state <- sample(LETTERS[1:4], nrow(d), replace=TRUE)
propsTrans(state ~ time + id, data=d)

pt1 <- data.frame(pt=1, day=0:3,
  status=c('well', 'well', 'sick', 'very sick'))
pt2 <- data.frame(pt=2, day=c(1,2,4,6),
  status=c('sick', 'very sick', 'coma', 'death'))
pt3 <- data.frame(pt=3, day=1:5,
  status=c('sick', 'very sick', 'sick', 'very sick', 'discharged'))
pt4 <- data.frame(pt=4, day=c(1:4, 10),
  status=c('well', 'sick', 'very sick', 'well', 'discharged'))
d <- rbind(pt1, pt2, pt3, pt4)
d$status <- factor(d$status, c('discharged', 'well', 'sick',
  'very sick', 'coma', 'death'))

label(d$day) <- 'Day'
require(ggplot2)
multEventChart(status ~ day + pt, data=d,
  absorb=c('death', 'discharged'),
  colorTitle='Status', sortbylast=TRUE) +
  theme_classic() +
  theme(legend.position='bottom')

```

---

princmp

*princmp*


---

## Description

Enhanced Output for Principal and Sparse Principal Components

## Usage

```

princmp(
  formula,
  data = environment(formula),
  method = c("regular", "sparse"),
  k = min(5, p - 1),
  kapprox = min(5, k),
  cor = TRUE,
  sw = FALSE,
  nvmax = 5
)

```

## Arguments

`formula` a formula with no left hand side, or a numeric matrix

data	a data frame or table. By default variables come from the calling environment.
method	specifies whether to use regular or sparse principal components are computed
k	the number of components to plot, display, and return
kapprox	the number of components to approximate with stepwise regression when sw=TRUE
cor	set to FALSE to compute PCs on the original data scale, which is useful if all variables have the same units of measurement
sw	set to TRUE to run stepwise regression PC prediction/approximation
nvmax	maximum number of predictors to allow in stepwise regression PC approximations

### Details

Expands any categorical predictors into indicator variables, and calls `princomp` (if `method='regular'` (the default)) or `sPCAgrid` in the `pcaPP` package (`method='sparse'`) to compute lasso-penalized sparse principal components. By default all variables are first scaled by their standard deviation after observations with any NAs on any variables in `formula` are removed. Loadings of standardized variables, and if `orig=TRUE` loadings on the original data scale are printed. If `p1=TRUE` a scree plot is drawn with text added to indicate cumulative proportions of variance explained. If `sw=TRUE`, the `leaps` package `regsubsets` function is used to approximate the PCs using forward stepwise regression with the original variables as individual predictors.

A `print` method prints the results and a `plot` method plots the scree plot of variance explained.

### Value

a list of class `princomp` with elements `scores`, a `k`-column matrix with principal component scores, with NAs when the input data had an NA, and other components useful for printing and plotting. If `k=1` `scores` is a vector. Other components include `vars` (vector of variances explained), `method`, `k`.

### Author(s)

Frank Harrell

---

`print.char.list`      *prints a list of lists in a visually readable format.*

---

### Description

Takes a list that is composed of other lists and matrixes and prints it in a visually readable format.

**Usage**

```
## S3 method for class 'char.list'
print(x, ..., hsep = c("|"), vsep = c("-"), csep = c("+"), print.it = TRUE,
      rowname.halign = c("left", "centre", "right"),
      rowname.valign = c("top", "centre", "bottom"),
      colname.halign = c("centre", "left", "right"),
      colname.valign = c("centre", "top", "bottom"),
      text.halign = c("right", "centre", "left"),
      text.valign = c("top", "centre", "bottom"),
      rowname.width, rowname.height,
      min.colwidth = .Options$digits, max.rowheight = NULL,
      abbreviate.dimnames = TRUE, page.width = .Options$width,
      colname.width, colname.height, prefix.width,
      superprefix.width = prefix.width)
```

**Arguments**

x	list object to be printed
...	place for extra arguments to reside.
hsep	character used to separate horizontal fields
vsep	character used to separate vertical fields
csep	character used where horizontal and vertical separators meet.
print.it	should the value be printed to the console or returned as a string.
rowname.halign	horizontal justification of row names.
rowname.valign	vertical justification of row names.
colname.halign	horizontal justification of column names.
colname.valign	vertical justification of column names.
text.halign	horizontal justification of cell text.
text.valign	vertical justification of cell text.
rowname.width	minimum width of row name strings.
rowname.height	minimum height of row name strings.
min.colwidth	minimum column width.
max.rowheight	maximum row height.
abbreviate.dimnames	should the row and column names be abbreviated.
page.width	width of the page being printed on.
colname.width	minimum width of the column names.
colname.height	minimum height of the column names
prefix.width	maximum width of the rowname columns
superprefix.width	maximum width of the super rowname columns



**Value**

String that formatted table of the list object.

**Author(s)**

Charles Dupont

---

print.char.matrix      *Function to print a matrix with stacked cells*

---

**Description**

Prints a dataframe or matrix in stacked cells. Line break characters in a matrix element will result in a line break in that cell, but tab characters are not supported.

**Usage**

```
## S3 method for class 'char.matrix'
print(x, file = "", col.name.align = "cen", col.txt.align = "right",
      cell.align = "cen", hsep = "|", vsep = "-", csep = "+", row.names = TRUE,
      col.names = FALSE, append = FALSE,
      top.border = TRUE, left.border = TRUE, ...)
```

**Arguments**

x	a matrix or dataframe
file	name of file if file output is desired. If left empty, output will be to the screen
col.name.align	if column names are used, they can be aligned right, left or centre. Default "cen" results in names centred between the sides of the columns they name. If the width of the text in the columns is less than the width of the name, col.name.align will have no effect. Other options are "right" and "left".
col.txt.align	how character columns are aligned. Options are the same as for col.name.align with no effect when the width of the column is greater than its name.
cell.align	how numbers are displayed in columns
hsep	character string to use as horizontal separator, i.e. what separates columns
vsep	character string to use as vertical separator, i.e. what separates rows. Length cannot be more than one.
csep	character string to use where vertical and horizontal separators cross. If hsep is more than one character, csep will need to be the same length. There is no provision for multiple vertical separators
row.names	logical: are we printing the names of the rows?
col.names	logical: are we printing the names of the columns?
append	logical: if file is not "", are we appending to the file or overwriting?
top.border	logical: do we want a border along the top above the columns?
left.border	logical: do we want a border along the left of the first column?
...	unused

**Details**

If any column of `x` is a mixture of character and numeric, the distinction between character and numeric columns will be lost. This is especially so if the matrix is of a form where you would not want to print the column names, the column information being in the rows at the beginning of the matrix.

Row names, if not specified in the making of the matrix will simply be numbers. To prevent printing them, set `row.names = FALSE`.

**Value**

No value is returned. The matrix or dataframe will be printed to file or to the screen.

**Author(s)**

Patrick Connolly <p.connolly@hortresearch.co.nz>

**See Also**

`write`, `write.table`

**Examples**

```
data(HairEyeColor)
print.char.matrix(HairEyeColor[ , , "Male"], col.names = TRUE)
print.char.matrix(HairEyeColor[ , , "Female"], col.txt.align = "left", col.names = TRUE)

z <- rbind(c("", "N", "y"),
           c("[ 1.34,40.3)\n[40.30,48.5)\n[48.49,58.4)\n[58.44,87.8]",
             " 50\n 50\n 50\n 50",
             "0.530\n0.489\n0.514\n0.507"),
           c("female\nmale", " 94\n106", "0.552\n0.473" ),
           c("", "200", "0.510"))
dimnames(z) <- list(c("", "age", "sex", "Overall"),NULL)

print.char.matrix(z)
```

---

`print.princmp`

*print.princmp*

---

**Description**

Print Results of `princmp`

**Usage**

```
## S3 method for class 'princmp'
print(x, which = c("none", "standardized", "original", "both"), k = x$k, ...)
```

**Arguments**

x	results of princmp
which	specifies which loadings to print, the default being 'none' and other values being 'standardized', 'original', or 'both'
k	number of components to show, defaults to k specified to princmp
...	unused

**Details**

Simple print method for [princmp\(\)](#)

**Value**

nothing

**Author(s)**

Frank Harrell

---

printL

*printL*

---

**Description**

Print an object or a named list of objects. When multiple objects are given, their names are printed before their contents. When an object is a vector that is not longer than `maxoneline` and its elements are not named, all the elements will be printed on one line separated by commas. When `dec` is given, numeric vectors or numeric columns of data frames or data tables are rounded to the nearest `dec` before printing. This function is especially helpful when printing objects in a Quarto or RMarkdown document and the code is not currently being shown to place the output in context.

**Usage**

```
printL(..., dec = NULL, maxoneline = 5)
```

**Arguments**

...	any number of objects to print()
dec	optional decimal places to the right of the decimal point for rounding
maxoneline	controls how many elements may be printed on a single line for vector objects

**Value**

nothing

**Author(s)**

Frank Harrell

**See Also**[prn\(\)](#)**Examples**

```
w <- pi + 1 : 2
printL(w=w)
printL(w, dec=3)
printL('this is it'=c(pi, pi, 1, 2),
      yy=pi,
      z=data.frame(x=pi+1:2, y=3:4, z=c('a', 'b')),
      qq=1:10,
      dec=4)
```

---

prnz

*Print and Object with its Name*

---

**Description**

Prints an object with its name and with an optional descriptive text string. This is useful for annotating analysis output files and for debugging.

**Usage**

```
prn(x, txt, file)
```

**Arguments**

x	any object
txt	optional text string
file	optional file name. By default, writes to console. append=TRUE is assumed.

**Side Effects**

prints

**See Also**

[print](#), [cat](#), [printL](#)

**Examples**

```
x <- 1:5
prn(x)
# prn(fit, 'Full Model Fit')
```

---

prselect                      *Selectively Print Lines of a Text Vector*

---

### Description

Given one or two regular expressions or exact text matches, removes elements of the input vector that match these specifications. Omitted lines are replaced by `...`. This is useful for selectively suppressing some of the printed output of R functions such as regression fitting functions, especially in the context of making statistical reports using Sweave or Odfweave.

### Usage

```
prselect(x, start = NULL, stop = NULL, i = 0, j = 0, pr = TRUE)
```

### Arguments

x	input character vector
start	text or regular expression to look for starting line to omit. If omitted, deletions start at the first line.
stop	text or regular expression to look for ending line to omit. If omitted, deletions proceed until the last line.
i	increment in number of first line to delete after match is found
j	increment in number of last line to delete after match is found
pr	set to FALSE to suppress printing

### Value

an invisible vector of retained lines of text

### Author(s)

Frank Harrell

### See Also

[Sweave](#)

### Examples

```
x <- c('the', 'cat', 'ran', 'past', 'the', 'dog')
prselect(x, 'big', 'bad')      # omit nothing- no match
prselect(x, 'the', 'past')    # omit first 4 lines
prselect(x, 'the', 'junk')    # omit nothing- no match for stop
prselect(x, 'ran', 'dog')     # omit last 4 lines
prselect(x, 'cat')            # omit lines 2-
prselect(x, 'cat', i=1)       # omit lines 3-
prselect(x, 'cat', 'past')    # omit lines 2-4
```

```

prselect(x,'cat','past',j=1) # omit lines 2-5
prselect(x,'cat','past',j=-1)# omit lines 2-3
prselect(x,'t$','dog')      # omit lines 2-6; t must be at end

# Example for Sweave: run a regression analysis with the rms package
# then selectively output only a portion of what print.ols prints.
# (Thanks to \email{romain.francois@dbmail.com})
# <<z,eval=FALSE,echo=T>>=
# library(rms)
# y <- rnorm(20); x1 <- rnorm(20); x2 <- rnorm(20)
# ols(y ~ x1 + x2)
# <<echo=F>>=
# z <- capture.output( {
# <<z>>
#   } )
# prselect(z, 'Residuals:') # keep only summary stats; or:
# prselect(z, stop='Coefficients', j=-1) # keep coefficients, rmse, R^2; or:
# prselect(z, 'Coefficients', 'Residual standard error', j=-1) # omit coef
# @

```

---

pstamp

*Date/Time/Directory Stamp the Current Plot*


---

## Description

Date-time stamp the current plot in the extreme lower right corner. Optionally add the current working directory and arbitrary other text to the stamp.

## Usage

```
pstamp(txt, pwd = FALSE, time. = TRUE)
```

## Arguments

txt	an optional single text string
pwd	set to TRUE to add the current working directory name to the stamp
time.	set to FALSE to use the date without the time

## Details

Certain functions are not supported for S-Plus under Windows. For R, results may not be satisfactory if `par(mfrow=)` is in effect.

## Author(s)

Frank Harrell

**Examples**

```
plot(1:20)
pstamp(pwd=TRUE, time=FALSE)
```

---

qcrypt

*qcrypt*


---

**Description**

Store and Encrypt R Objects or Files or Read and Decrypt Them

**Usage**

```
qcrypt(obj, base, service = "R-keyring-service", file)
```

**Arguments**

obj	an R object to write to disk and encrypt (if base is specified) or the base file name to read and unencrypted (if base is not specified). Not used when file is given.
base	base file name when creating a file. Not used when file is given.
service	a fairly arbitrary keyring service name. The default is almost always OK unless you need to use different passwords for different files.
file	full name of file to encrypt or decrypt

**Details**

qcrypt is used to protect sensitive information on a user's computer or when transmitting a copy of the file to another R user. Unencrypted information only exists for a moment, and the encryption password does not appear in the user's script but instead is managed by the keyring package to remember the password across R sessions, and the getPass package, which pops up a password entry window and does not allow the password to be visible. The password is requested only once, except perhaps when the user logs out of their operating system session or reboots.

The keyring can be bypassed and the password entered in a popup window by specifying `service=NA`. This is the preferred approach when sending an encrypted file to a user on a different computer.

qcrypt writes R objects to disk in a temporary file using the qs package `qsave` function. The file is quickly encrypted using the `safer` package, and the temporary unencrypted qs file is deleted. When reading an encrypted file the process is reversed.

To save an object in an encrypted file, specify the object as the first argument `obj` and specify a base file name as a character string in the second argument `base`. The full qs file name will be of the form `base.qs.encrypted` in the user's current working directory. To unencrypt the file into a short-lived temporary file and use `qs::qread` to read it, specify the base file name as a character string with the first argument, and do not specify the base argument.

Alternatively, qcrypt can be used to encrypt or decrypt existing files of any type using the same password and keyring mechanism. The former is done by specifying `file` that does not end in

'.encrypted' and the latter is done by ending file with '.encrypted'. When file does not contain a path it is assumed to be in the current working directory. When a file is encrypted the original file is removed. Files are decrypted into a temporary directory created by `tempdir()`, with the name of the file being the value of file with '.encrypted' removed.

Interactive password provision works when running R, Rscript, RStudio, or Quarto but does not work when running R CMD BATCH. `getPass` fails under RStudio on Macs.

See [R Workflow](#) for more information.

## Value

(invisibly) the full encrypted file name if writing the file, or the restored R object if reading the file. When decrypting a general file with `file=`, the returned value is the full path to a temporary file containing the decrypted data.

## Author(s)

Frank Harrell

## Examples

```
## Not run:
# Suppose x is a data.table or data.frame
# The first time qcrypt is run with a service a password will
# be requested. It will be remembered across sessions thanks to
# the keyring package
qcrypt(x, 'x') # creates x.qs.encrypted in current working directory
x <- qcrypt('x') # unencrypts x.qs.encrypted into a temporary
# directory, uses qs::qread to read it, and
# stores the result in x
# Encrypt a general file using a different password
qcrypt(file='report.pdf', service='pdfkey')
# Decrypt that file
fi <- qcrypt(file='report.pdf.encrypted', service='pdfkey')
fi contains the full unencrypted file name which is in a temporary directory
# Encrypt without using a keyring
qcrypt(x, 'x', service=NA)
x <- qcrypt('x', service=NA)

## End(Not run)
```

---

qrxcenter

*qrxcenter*

---

## Description

Mean-center a data matrix and QR transform it



**Usage**

```
qrxcenter(x, ...)
```

**Arguments**

`x` a numeric matrix or vector with at least 2 rows  
`...` passed to `base::qr()`

**Details**

For a numeric matrix `x` (or a numeric vector that is automatically changed to a one-column matrix), computes column means and subtracts them from `x` columns, and passes this matrix to `base::qr()` to orthogonalize columns. Columns of the transformed `x` are negated as needed so that original directions are preserved (which are arbitrary with QR decomposition). Instead of the default `qr` operation for which sums of squares of column values are 1.0, `qrxcenter` makes all the transformed columns have standard deviation of 1.0.

**Value**

a list with components `x` (transformed data matrix), `R` (the matrix that can be used to transform raw `x` and to transform regression coefficients computed on transformed `x` back to the original space), `Ri` (transforms transformed `x` back to original scale except for `xbar`), and `xbar` (vector of means of original `x` columns')

**Examples**

```
set.seed(1)
age <- 1:10
country <- sample(c('Slovenia', 'Italy', 'France'), 10, TRUE)
x <- model.matrix(~ age + country)[, -1]
x
w <- qrxcenter(x)
w
# Reproduce w$x
sweep(x, 2, w$xbar) %*% w$R
# Reproduce x from w$x
sweep(w$x %*% w$Ri, 2, w$xbar, FUN='+')
# See also https://hbiostat.org/r/examples/gtrans/gtrans#sec-splinebasis
```

---

r2describe

*r2describe*


---

**Description**

Summarize Strength of Relationships Using R-Squared From Linear Regression

**Usage**

```
r2describe(x, nvmax = 10)
```

**Arguments**

x                    numeric matrix with 2 or more columns  
 nvmax                maximum number of columns of x to use in predicting a given column

**Details**

Function to use `leaps::regsubsets()` to briefly describe which variables more strongly predict another variable. Variables are in a numeric matrix and are assumed to be transformed so that relationships are linear (e.g., using `redun()` or `transcan()`.)

**Value**

nothing

**Author(s)**

Frank Harrell

**Examples**

```
## Not run:
r <- redun(...)
r2describe(r$scores)

## End(Not run)
```

---

R2Measures

*R2Measures*


---

**Description**

Generalized R<sup>2</sup> Measures

**Usage**

```
R2Measures(lr, p, n, ess = NULL, padj = 1)
```

**Arguments**

lr                    likelihood ratio chi-square statistic  
 p                    number of non-intercepts in the model that achieved lr  
 n                    raw number of observations  
 ess                  if a single number, is the effective sample size. If a vector of numbers is assumed to be the frequency tabulation of all distinct values of the outcome variable, from which the effective sample size is computed.  
 padj                set to 2 to use the classical adjusted R<sup>2</sup> penalty, 1 (the default) to subtract p from lr

## Details

Computes various generalized  $R^2$  measures related to the Maddala-Cox-Snell (MCS)  $R^2$  for regression models fitted with maximum likelihood. The original MCS  $R^2$  is labeled R2 in the result. This measure uses the raw sample size  $n$  and does not penalize for the number of free parameters, so it can be rewarded for overfitting. A measure adjusted for the number of fitted regression coefficients  $p$  uses the analogy to  $R^2$  in linear models by computing  $1 - \exp(-lr / n) * (n-1)/(n-p-1)$  if  $padj=2$ , which is approximately  $1 - \exp(-(lr - p) / n)$ , the version used if  $padj=1$  (the default). The latter measure is appealing because the expected value of the likelihood ratio chi-square statistic  $lr$  is  $p$  under the global null hypothesis of no predictors being associated with the response variable. See <https://hbiostat.org/bib/r2.html> for more details.

It is well known that in logistic regression the MCS  $R^2$  cannot achieve a value of 1.0 even with a perfect model, which prompted Nagelkerke to divide the  $R^2$  measure by its maximum attainable value. This is not necessarily the best recalibration of  $R^2$  throughout its range. An alternative is to use the formulas above but to replace the raw sample size  $n$  with the effective sample size, which for data with many ties can be significantly lower than the number of observations. As used in the `popower()` and `describe()` functions, in the context of a Wilcoxon test or the proportional odds model, the effective sample size is  $n * (1 - f)$  where  $f$  is the sums of cubes of the proportion of observations at each distinct value of the response variable. Whitehead derived this from an approximation to the variance of a log odds ratio in a proportional odds model. To obtain  $R^2$  measures using the effective sample size, either provide `ess` as a single number specifying the effective sample size, or specify a vector of frequencies of distinct  $Y$  values from which the effective sample size will be computed. In the context of survival analysis, the single number effective sample size you may wish to specify is the number of uncensored observations. This is exactly correct when estimating the hazard rate from a simple exponential distribution or when using the Cox PH/log-rank test. For failure time distributions with a very high early hazard, censored observations contain enough information that the effective sample size is greater than the number of events. See Benedetti et al, 1982.

If the effective sample size equals the raw sample size, measures involving the effective sample size are set to NA.

## Value

named vector of R2 measures. The notation for results is  $R^2(p, n)$  where the  $p$  component is empty for unadjusted estimates and  $n$  is the sample size used (actual sample size for first measures, effective sample size for remaining ones). For indexes that are not adjusted, only  $n$  appears.

## Author(s)

Frank Harrell

## References

Smith TJ and McKenna CM (2013): A comparison of logistic regression pseudo  $R^2$  indices. Multiple Linear Regression Viewpoints 39:17-26. [https://www.glmj.org/archives/articles/Smith\\_v39n2.pdf](https://www.glmj.org/archives/articles/Smith_v39n2.pdf)

Benedetti JK, et al (1982): Effective sample size for tests of censored survival data. Biometrika 69:343-349.

Mittlbock M, Schemper M (1996): Explained variation for logistic regression. *Stat in Med* 15:1987-1997.

Date, S: R-squared, adjusted R-squared and pseudo R-squared. <https://timeseriesreasoning.com/contents/r-squared-adjusted-r-squared-pseudo-r-squared/>

UCLA: What are pseudo R-squareds? <https://stats.oarc.ucla.edu/other/mult-pkg/faq/general/faq-what-are-pseudo-r-squareds/>

Allison P (2013): What's the beset R-squared for logistic regression? <https://statisticalhorizons.com/r2logistic/>

Menard S (2000): Coefficients of determination for multiple logistic regression analysis. *The Am Statistician* 54:17-24.

Whitehead J (1993): Sample size calculations for ordered categorical data. *Stat in Med* 12:2257-2271. See errata (1994) 13:871 and letter to the editor by Julious SA, Campbell MJ (1996) 15:1065-1066 showing that for 2-category Y the Whitehead sample size formula agrees closely with the usual formula for comparing two proportions.

## Examples

```
x <- c(rep(0, 50), rep(1, 50))
y <- x
# f <- lrm(y ~ x)
# f # Nagelkerke R^2=1.0
# lr <- f$stats['Model L.R. ']
# 1 - exp(- lr / 100) # Maddala-Cox-Snell (MCS) 0.75
lr <- 138.6267 # manually so don't need rms package

R2Measures(lr, 1, 100, c(50, 50)) # 0.84 Effective n=75
R2Measures(lr, 1, 100, 50) # 0.94
# MCS requires unreasonable effective sample size = minimum outcome
# frequency to get close to the 1.0 that Nagelkerke R^2 achieves
```

---

rcorr

*Matrix of Correlations and P-values*

---

## Description

rcorr Computes a matrix of Pearson's  $r$  or Spearman's  $\rho$  rank correlation coefficients for all possible pairs of columns of a matrix. Missing values are deleted in pairs rather than deleting all rows of  $x$  having any missing variables. Ranks are computed using efficient algorithms (see reference 2), using midranks for ties.

## Usage

```
rcorr(x, y, type=c("pearson", "spearman"))

## S3 method for class 'rcorr'
print(x, ...)
```

**Arguments**

x	a numeric matrix with at least 5 rows and at least 2 columns (if y is absent). For print, x is an object produced by rcorr.
y	a numeric vector or matrix which will be concatenated to x. If y is omitted for rcorr, x must be a matrix.
type	specifies the type of correlations to compute. Spearman correlations are the Pearson linear correlations computed on the ranks of non-missing elements, using midranks for ties.
...	argument for method compatibility.

**Details**

Uses midranks in case of ties, as described by Hollander and Wolfe. P-values are approximated by using the t or F distributions.

**Value**

rcorr returns a list with elements r, the matrix of correlations, n the matrix of number of observations used in analyzing each pair of variables, P, the asymptotic P-values, and type. Pairs with fewer than 2 non-missing values have the r values set to NA. The diagonals of n are the number of non-NAs for the single variable corresponding to that row and column.

**Author(s)**

Frank Harrell  
Department of Biostatistics  
Vanderbilt University  
<fh@fharrell.com>

**References**

Hollander M. and Wolfe D.A. (1973). Nonparametric Statistical Methods. New York: Wiley.  
Press WH, Flannery BP, Teukolsky SA, Vetterling, WT (1988): Numerical Recipes in C. Cambridge: Cambridge University Press.

**See Also**

[hoeffd](#), [cor](#), [combine.levels](#), [varclus](#), [dotchart3](#), [impute](#), [chisq.test](#), [cut2](#).

**Examples**

```
x <- c(-2, -1, 0, 1, 2)
y <- c(4, 1, 0, 1, 4)
z <- c(1, 2, 3, 4, NA)
v <- c(1, 2, 3, 4, 5)
rcorr(cbind(x,y,z,v))
```

rcorr.cens

*Rank Correlation for Censored Data***Description**

Computes the  $c$  index and the corresponding generalization of Somers' Dxy rank correlation for a censored response variable. Also works for uncensored and binary responses, although its use of all possible pairings makes it slow for this purpose. Dxy and  $c$  are related by  $Dxy = 2(c - 0.5)$ .

rcorr.cens handles one predictor variable. rcorr.cens computes rank correlation measures separately by a series of predictors. In addition, rcorr.cens has a rough way of handling categorical predictors. If a categorical (factor) predictor has two levels, it is converted to a numeric having values 1 and 2. If it has more than 2 levels, an indicator variable is formed for the most frequently level vs. all others, and another indicator for the second most frequent level and all others. The correlation is taken as the maximum of the two (in absolute value).

**Usage**

```
rcorr.cens(x, S, outx=FALSE)

## S3 method for class 'formula'
rcorr.cens(formula, data=NULL, subset=NULL,
            na.action=na.retain, exclude.imputed=TRUE, outx=FALSE,
            ...)
```

**Arguments**

x	a numeric predictor variable
S	an Surv object or a vector. If a vector, assumes that every observation is uncensored.
outx	set to TRUE to not count pairs of observations tied on x as a relevant pair. This results in a Goodman–Kruskal gamma type rank correlation.
formula	a formula with a Surv object or a numeric vector on the left-hand side
data, subset, na.action	the usual options for models. Default for na.action is to retain all values, NA or not, so that NAs can be deleted in only a pairwise fashion.
exclude.imputed	set to FALSE to include imputed values (created by impute) in the calculations.
...	extra arguments passed to <a href="#">biVar</a> .

**Value**

rcorr.cens returns a vector with the following named elements: C Index, Dxy, S.D., n, missing, uncensored, Relevant Pairs, Concordant, and Uncertain

n	number of observations not missing on any input variables
---	---

missing	number of observations missing on x or S
relevant	number of pairs of non-missing observations for which S could be ordered
concordant	number of relevant pairs for which x and S are concordant.
uncertain	number of pairs of non-missing observations for which censoring prevents classification of concordance of x and S.

`rcorr.cens`.formula returns an object of class `biVar` which is documented with the `biVar` function.

### Author(s)

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>

### References

Newson R: Confidence intervals for rank statistics: Somers' D and extensions. *Stata Journal* 6:309-334; 2006.

### See Also

[concordance](#), [somers2](#), [biVar](#), [rcorrrp.cens](#)

### Examples

```
set.seed(1)
x <- round(rnorm(200))
y <- rnorm(200)
rcorr.cens(x, y, outx=TRUE) # can correlate non-censored variables
library(survival)
age <- rnorm(400, 50, 10)
bp <- rnorm(400, 120, 15)
bp[1] <- NA
d.time <- rexp(400)
cens <- runif(400, .5, 2)
death <- d.time <= cens
d.time <- pmin(d.time, cens)
rcorr.cens(age, Surv(d.time, death))
r <- rcorrcens(Surv(d.time, death) ~ age + bp)
r
plot(r)

# Show typical 0.95 confidence limits for ROC areas for a sample size
# with 24 events and 62 non-events, for varying population ROC areas
# Repeat for 138 events and 102 non-events
set.seed(8)
par(mfrow=c(2,1))
for(i in 1:2) {
  n1 <- c(24,138)[i]
  n0 <- c(62,102)[i]
```

```

y <- c(rep(0,n0), rep(1,n1))
deltas <- seq(-3, 3, by=.25)
C <- se <- deltas
j <- 0
for(d in deltas) {
  j <- j + 1
  x <- c(rnorm(n0, 0), rnorm(n1, d))
  w <- rcorr.cens(x, y)
  C[j] <- w['C Index']
  se[j] <- w['S.D.']/2
}
low <- C-1.96*se; hi <- C+1.96*se
print(cbind(C, low, hi))
errbar(deltas, C, C+1.96*se, C-1.96*se,
       xlab='True Difference in Mean X',
       ylab='ROC Area and Approx. 0.95 CI')
title(paste('n1=',n1,' n0=',n0,sep=''))
abline(h=.5, v=0, col='gray')
true <- 1 - pnorm(0, deltas, sqrt(2))
lines(deltas, true, col='blue')
}
par(mfrow=c(1,1))

```

---

rccorr.cens

*Rank Correlation for Paired Predictors with a Possibly Censored Response, and Integrated Discrimination Index*


---

## Description

Computes U-statistics to test for whether predictor X1 is more concordant than predictor X2, extending `rccorr.cens`. For `method=1`, estimates the fraction of pairs for which the x1 difference is more impressive than the x2 difference. For `method=2`, estimates the fraction of pairs for which x1 is concordant with S but x2 is not.

For binary responses the function `improveProb` provides several assessments of whether one set of predicted probabilities is better than another, using the methods describe in *Pencina et al (2007)*. This involves NRI and IDI to test for whether predictions from model x1 are significantly different from those obtained from predictions from model x2. This is a distinct improvement over comparing ROC areas, sensitivity, or specificity.

## Usage

```
rccorr.cens(x1, x2, S, outx=FALSE, method=1)
```

```
improveProb(x1, x2, y)
```

```
## S3 method for class 'improveProb'
print(x, digits=3, conf.int=.95, ...)
```



**Arguments**

x1	first predictor (a probability, for improveProb)
x2	second predictor (a probability, for improveProb)
S	a possibly right-censored <a href="#">Surv</a> object. If S is a vector instead, it is converted to a <a href="#">Surv</a> object and it is assumed that no observations are censored.
outx	set to TRUE to exclude pairs tied on x1 or x2 from consideration
method	see above
y	a binary 0/1 outcome variable
x	the result from improveProb
digits	number of significant digits for use in printing the result of improveProb
conf.int	level for confidence limits
...	unused

**Details**

If x1,x2 represent predictions from models, these functions assume either that you are using a separate sample from the one used to build the model, or that the amount of overfitting in x1 equals the amount of overfitting in x2. An example of the latter is giving both models equal opportunity to be complex so that both models have the same number of effective degrees of freedom, whether a predictor was included in the model or was screened out by a variable selection scheme.

Note that in the first part of their paper, *Pencina et al.* presented measures that required binning the predicted probabilities. Those measures were then replaced with better continuous measures that are implemented here.

**Value**

a vector of statistics for rccorrp.cens, or a list with class improveProb of statistics for improveProb:

n	number of cases
na	number of events
nb	number of non-events
pup.ev	mean of pairwise differences in probabilities for those with events and a pairwise difference of probabilities > 0
pup.ne	mean of pairwise differences in probabilities for those without events and a pairwise difference of probabilities > 0
pdown.ev	mean of pairwise differences in probabilities for those with events and a pairwise difference of probabilities > 0
pdown.ne	mean of pairwise differences in probabilities for those without events and a pairwise difference of probabilities > 0
nri	Net Reclassification Index = $(pup.ev - pdown.ev) - (pup.ne - pdown.ne)$
se.nri	standard error of NRI
z.nri	Z score for NRI

nri.ev	Net Reclassification Index = $pup.ev - pdown.ev$
se.nri.ev	SE of NRI of events
z.nri.ev	Z score for NRI of events
nri.ne	Net Reclassification Index = $pup.ne - pdown.ne$
se.nri.ne	SE of NRI of non-events
z.nri.ne	Z score for NRI of non-events
improveSens	improvement in sensitivity
improveSpec	improvement in specificity
idi	Integrated Discrimination Index
se.idi	SE of IDI
z.idi	Z score of IDI

**Author(s)**

Frank Harrell  
 Department of Biostatistics, Vanderbilt University  
 <fh@fharrell.com>

Scott Williams  
 Division of Radiation Oncology  
 Peter MacCallum Cancer Centre, Melbourne, Australia  
 <scott.williams@petermac.org>

**References**

- Pencina MJ, D'Agostino Sr RB, D'Agostino Jr RB, Vasan RS (2008): Evaluating the added predictive ability of a new marker: From area under the ROC curve to reclassification and beyond. *Stat in Med* 27:157-172. DOI: 10.1002/sim.2929
- Pencina MJ, D'Agostino Sr RB, D'Agostino Jr RB, Vasan RS: Rejoinder: Comments on Integrated discrimination and net reclassification improvements-Practical advice. *Stat in Med* 2007; DOI: 10.1002/sim.3106
- Pencina MJ, D'Agostino RB, Steyerberg EW (2011): Extensions of net reclassification improvement calculations to measure usefulness of new biomarkers. *Stat in Med* 30:11-21; DOI: 10.1002/sim.4085

**See Also**

[rccorr.cens](#), [somers2](#), [Surv](#), [val.prob](#), [concordance](#)

**Examples**

```
set.seed(1)
library(survival)

x1 <- rnorm(400)
x2 <- x1 + rnorm(400)
d.time <- rexp(400) + (x1 - min(x1))
cens <- runif(400, .5, 2)
```

```

death <- d.time <= cens
d.time <- pmin(d.time, cens)
rcorrp.cens(x1, x2, Surv(d.time, death))
#rcorrp.cens(x1, x2, y) ## no censoring

set.seed(1)
x1 <- runif(1000)
x2 <- runif(1000)
y <- sample(0:1, 1000, TRUE)
rcorrp.cens(x1, x2, y)
improveProb(x1, x2, y)

```

rcspline.eval

*Restricted Cubic Spline Design Matrix***Description**

Computes matrix that expands a single variable into the terms needed to fit a restricted cubic spline (natural spline) function using the truncated power basis. Two normalization options are given for somewhat reducing problems of ill-conditioning. The antiderivative function can be optionally created. If knot locations are not given, they will be estimated from the marginal distribution of  $x$ .

**Usage**

```

rcspline.eval(x, knots, nk=5, inclx=FALSE, knots.only=FALSE,
              type="ordinary", norm=2, rpm=NULL, pc=FALSE,
              fractied=0.05)

```

**Arguments**

<code>x</code>	a vector representing a predictor variable
<code>knots</code>	knot locations. If not given, knots will be estimated using default quantiles of $x$ . For 3 knots, the outer quantiles used are 0.10 and 0.90. For 4-6 knots, the outer quantiles used are 0.05 and 0.95. For $nk > 6$ , the outer quantiles are 0.025 and 0.975. The knots are equally spaced between these on the quantile scale. For fewer than 100 non-missing values of $x$ , the outer knots are the 5th smallest and largest $x$ .
<code>nk</code>	number of knots. Default is 5. The minimum value is 3.
<code>inclx</code>	set to TRUE to add $x$ as the first column of the returned matrix
<code>knots.only</code>	return the estimated knot locations but not the expanded matrix
<code>type</code>	"ordinary" to fit the function, "integral" to fit its anti-derivative.
<code>norm</code>	'0' to use the terms as originally given by <i>Devlin and Weeks (1986)</i> , '1' to normalize non-linear terms by the cube of the spacing between the last two knots, '2' to normalize by the square of the spacing between the first and last knots (the default). <code>norm=2</code> has the advantage of making all nonlinear terms be on the $x$ -scale.

rpm	If given, any NAs in x will be replaced with the value rpm after estimating any knot locations.
pc	Set to TRUE to replace the design matrix with orthogonal (uncorrelated) principal components computed on the scaled, centered design matrix
fractied	If the fraction of observations tied at the lowest and/or highest values of x is greater than or equal to fractied, the algorithm attempts to use a different algorithm for knot finding based on quantiles of x after excluding the one or two values with excessive ties. And if the number of unique x values excluding these values is small, the unique values will be used as the knots. If the number of knots to use other than these exterior values is only one, that knot will be at the median of the non-extreme x. This algorithm is not used if any interior values of x also have a proportion of ties equal to or exceeding fractied.

### Value

If knots.only=TRUE, returns a vector of knot locations. Otherwise returns a matrix with x (if inclx=TRUE) followed by  $nk - 2$  nonlinear terms. The matrix has an attribute knots which is the vector of knots used. When pc is TRUE, an additional attribute is stored: pcparms, which contains the center and scale vectors and the rotation matrix.

### References

Devlin TF and Weeks BJ (1986): Spline functions for logistic regression modeling. Proc 11th Annual SAS Users Group Intl Conf, p. 646–651. Cary NC: SAS Institute, Inc.

### See Also

[ns](#), [rcspline.restate](#), [rcs](#)

### Examples

```
x <- 1:100
rcspline.eval(x, nk=4, inclx=TRUE)
#lrm.fit(rcspline.eval(age,nk=4,inclx=TRUE), death)
x <- 1:1000
attributes(rcspline.eval(x))
x <- c(rep(0, 744),rep(1,6), rep(2,4), rep(3,10),rep(4,2),rep(6,6),
  rep(7,3),rep(8,2),rep(9,4),rep(10,2),rep(11,9),rep(12,10),rep(13,13),
  rep(14,5),rep(15,5),rep(16,10),rep(17,6),rep(18,3),rep(19,11),rep(20,16),
  rep(21,6),rep(22,16),rep(23,17), 24, rep(25,8), rep(26,6),rep(27,3),
  rep(28,7),rep(29,9),rep(30,10),rep(31,4),rep(32,4),rep(33,6),rep(34,6),
  rep(35,4), rep(36,5), rep(38,6), 39, 39, 40, 40, 40, 41, 43, 44, 45)
attributes(rcspline.eval(x, nk=3))
attributes(rcspline.eval(x, nk=5))
u <- c(rep(0,30), 1:4, rep(5,30))
attributes(rcspline.eval(u))
```

rcspline.plot

*Plot Restricted Cubic Spline Function***Description**

Provides plots of the estimated restricted cubic spline function relating a single predictor to the response for a logistic or Cox model. The `rcspline.plot` function does not allow for interactions as do `lrm` and `cph`, but it can provide detailed output for checking spline fits. This function uses the `rcspline.eval`, `lrm.fit`, and Therneau's `coxph.fit` functions and plots the estimated spline regression and confidence limits, placing summary statistics on the graph. If there are no adjustment variables, `rcspline.plot` can also plot two alternative estimates of the regression function when `model="logistic"`: proportions or logit proportions on grouped data, and a nonparametric estimate. The nonparametric regression estimate is based on smoothing the binary responses and taking the logit transformation of the smoothed estimates, if desired. The smoothing uses `supsmu`.

**Usage**

```
rcspline.plot(x,y,model=c("logistic", "cox", "ols"), xrange, event, nk=5,
             knots=NULL, show=c("xbeta","prob"), adj=NULL, xlab, ylab,
             ylim, plim=c(0,1), plotcl=TRUE, showknots=TRUE, add=FALSE,
             subset, lty=1, noprint=FALSE, m, smooth=FALSE, bass=1,
             main="auto", statloc)
```

**Arguments**

<code>x</code>	a numeric predictor
<code>y</code>	a numeric response. For binary logistic regression, <code>y</code> should be either 0 or 1.
<code>model</code>	"logistic" or "cox". For "cox", uses the <code>coxph.fit</code> function with <code>method="efron"</code> argument set.
<code>xrange</code>	range for evaluating <code>x</code> , default is $f$ and $1-f$ quantiles of <code>x</code> , where $f = \frac{10}{\max(n,200)}$
<code>event</code>	event/censoring indicator if <code>model="cox"</code> . If <code>event</code> is present, <code>model</code> is assumed to be "cox"
<code>nk</code>	number of knots
<code>knots</code>	knot locations, default based on quantiles of <code>x</code> (by <code>rcspline.eval</code> )
<code>show</code>	"xbeta" or "prob" - what is plotted on <code>y</code> -axis
<code>adj</code>	optional matrix of adjustment variables
<code>xlab</code>	<code>x</code> -axis label, default is the "label" attribute of <code>x</code>
<code>ylab</code>	<code>y</code> -axis label, default is the "label" attribute of <code>y</code>
<code>ylim</code>	<code>y</code> -axis limits for logit or log hazard
<code>plim</code>	<code>y</code> -axis limits for probability scale
<code>plotcl</code>	plot confidence limits
<code>showknots</code>	show knot locations with arrows
<code>add</code>	add this plot to an already existing plot

subset	subset of observations to process, e.g. <code>sex == "male"</code>
lty	line type for plotting estimated spline function
noprint	suppress printing regression coefficients and standard errors
m	for <code>model="logistic"</code> , plot grouped estimates with triangles. Each group contains <code>m</code> ordered observations on <code>x</code> .
smooth	plot nonparametric estimate if <code>model="logistic"</code> and <code>adj</code> is not specified
bass	smoothing parameter (see <code>supsmu</code> )
main	main title, default is "Estimated Spline Transformation"
statloc	location of summary statistics. Default positioning by clicking left mouse button where upper left corner of statistics should appear. Alternative is "ll" to place below the graph on the lower left, or the actual <code>x</code> and <code>y</code> coordinates. Use "none" to suppress statistics.

### Value

list with components ('knots', 'x', 'xbeta', 'lower', 'upper') which are respectively the knot locations, design matrix, linear predictor, and lower and upper confidence limits

### Author(s)

Frank Harrell  
 Department of Biostatistics, Vanderbilt University  
 <fh@fharrell.com>

### See Also

[lrm](#), [cph](#), [rcspline.eval](#), [plot](#), [supsmu](#), [coxph.fit](#), [lrm.fit](#)

### Examples

```
#rcspline.plot(cad.dur, tvdlm, m=150)
#rcspline.plot(log10(cad.dur+1), tvdlm, m=150)
```

---

rcspline.restate

*Re-state Restricted Cubic Spline Function*

---

### Description

This function re-states a restricted cubic spline function in the un-linearly-restricted form. Coefficients for that form are returned, along with an R functional representation of this function and a LaTeX character representation of the function. `rcsplineFunction` is a fast function that creates a function to compute a restricted cubic spline function with given coefficients and knots, without reformatting the function to be pretty (i.e., into unrestricted form).

**Usage**

```
rcspline.restate(knots, coef,
                 type=c("ordinary", "integral"),
                 x="X", lx=nchar(x),
                 norm=2, columns=65, before="& &", after="\\",
                 begin="", nbegin=0, digits=max(8, .Options$digits))

rcsplineFunction(knots, coef, norm=2, type=c('ordinary', 'integral'))
```

**Arguments**

knots	vector of knots used in the regression fit
coef	vector of coefficients from the fit. If the length of coef is $k - 1$ , where $k$ is equal to the length(knots), the first coefficient must be for the linear term and remaining $k - 2$ coefficients must be for the constructed terms (e.g., from rcspline.eval). If the length of coef is $k$ , an intercept is assumed to be in the first element (or a zero is prepended to coef for rcsplineFunction).
type	The default is to represent the cubic spline function corresponding to the coefficients and knots. Set type = "integral" to instead represent its anti-derivative.
x	a character string to use as the variable name in the LaTeX expression for the formula.
lx	length of x to count with respect to columns. Default is length of character string contained by x. You may want to set lx smaller than this if it includes non-printable LaTeX commands.
norm	normalization that was used in deriving the original nonlinear terms used in the fit. See rcspline.eval for definitions.
columns	maximum number of symbols in the LaTeX expression to allow before inserting a newline ( <code>\</code> ) command. Set to a very large number to keep text all on one line.
before	text to place before each line of LaTeX output. Use <code>"&amp; &amp;"</code> for an equation array environment in LaTeX where you want to have a left-hand prefix e.g. <code>"f(X) &amp; = &amp;"</code> or using <code>"\lefteqn"</code> .
after	text to place at the end of each line of output.
begin	text with which to start the first line of output. Useful when adding LaTeX output to part of an existing formula
nbegin	number of columns of printable text in begin
digits	number of significant digits to write for coefficients and knots

**Value**

rcspline.restate returns a vector of coefficients. The coefficients are un-normalized and two coefficients are added that are linearly dependent on the other coefficients and knots. The vector of coefficients has four attributes. knots is a vector of knots, latex is a vector of text strings with the LaTeX representation of the formula. columns.used is the number of columns used in the output string since the last newline command. function is an R function, which is also return in character

string format as the text attribute. `rcsplineFunction` returns an R function with arguments `x` (a user-supplied numeric vector at which to evaluate the function), and some automatically-supplied other arguments.

### Author(s)

Frank Harrell  
 Department of Biostatistics, Vanderbilt University  
 <fh@fharrell.com>

### See Also

[rcspline.eval](#), [ns](#), [rcs](#), [latex](#), [Function.transcan](#)

### Examples

```
set.seed(1)
x <- 1:100
y <- (x - 50)^2 + rnorm(100, 0, 50)
plot(x, y)
xx <- rcspline.eval(x, inclx=TRUE, nk=4)
knots <- attr(xx, "knots")
coef <- lsfit(xx, y)$coef
options(digits=4)
# rcspline.restate must ignore intercept
w <- rcspline.restate(knots, coef[-1], x="{\\rm BP}")
# could also have used coef instead of coef[-1], to include intercept
cat(attr(w, "latex"), sep="\n")

xtrans <- eval(attr(w, "function"))
# This is an S function of a single argument
lines(x, coef[1] + xtrans(x), type="l")
# Plots fitted transformation

xtrans <- rcsplineFunction(knots, coef)
xtrans
lines(x, xtrans(x), col='blue')

#x <- blood.pressure
xx.simple <- cbind(x, pmax(x-knots[1],0)^3, pmax(x-knots[2],0)^3,
                  pmax(x-knots[3],0)^3, pmax(x-knots[4],0)^3)
pred.value <- coef[1] + xx.simple %*% w
plot(x, pred.value, type='l') # same as above
```



---

redun *Redundancy Analysis*


---

**Description**

Uses flexible parametric additive models (see [areg](#) and its use of regression splines), or alternatively to run a regular regression after replacing continuous variables with ranks, to determine how well each variable can be predicted from the remaining variables. Variables are dropped in a stepwise fashion, removing the most predictable variable at each step. The remaining variables are used to predict. The process continues until no variable still in the list of predictors can be predicted with an  $R^2$  or adjusted  $R^2$  of at least  $r2$  or until dropping the variable with the highest  $R^2$  (adjusted or ordinary) would cause a variable that was dropped earlier to no longer be predicted at least at the  $r2$  level from the now smaller list of predictors.

There is also an option `qrnk` to expand each variable into two columns containing the rank and square of the rank. Whenever ranks are used, they are computed as fractional ranks for numerical reasons.

**Usage**

```
redun(formula, data=NULL, subset=NULL, r2 = 0.9,
      type = c("ordinary", "adjusted"), nk = 3, tlinear = TRUE,
      rank=qrnk, qrnk=FALSE,
      allcat=FALSE, minfreq=0, iterm=FALSE, pc=FALSE, pr = FALSE, ...)
## S3 method for class 'redun'
print(x, digits=3, long=TRUE, ...)
```

**Arguments**

<code>formula</code>	a formula. Enclose a variable in <code>I()</code> to force linearity. Alternately, can be a numeric matrix, in which case the data are not run through <code>dataframeReduce</code> . This is useful when running the data through <code>transcan</code> first for nonlinearly transforming the data.
<code>data</code>	a data frame, which must be omitted if <code>formula</code> is a matrix
<code>subset</code>	usual subsetting expression
<code>r2</code>	ordinary or adjusted $R^2$ cutoff for redundancy
<code>type</code>	specify "adjusted" to use adjusted $R^2$
<code>nk</code>	number of knots to use for continuous variables. Use <code>nk=0</code> to force linearity for all variables.
<code>tlinear</code>	set to <code>FALSE</code> to allow a variable to be automatically nonlinearly transformed (see <a href="#">areg</a> ) while being predicted. By default, only continuous variables on the right hand side (i.e., while they are being predictors) are automatically transformed, using regression splines. Estimating transformations for target (dependent) variables causes more overfitting than doing so for predictors.
<code>rank</code>	set to <code>TRUE</code> to replace non-categorical variables with ranks before running the analysis. This causes <code>nk</code> to be set to zero.

qrnk	set to TRUE to also include squares of ranks to allow for non-monotonic transformations
allcat	set to TRUE to ensure that all categories of categorical variables having more than two categories are redundant (see details below)
minfreq	For a binary or categorical variable, there must be at least two categories with at least minfreq observations or the variable will be dropped and not checked for redundancy against other variables. minfreq also specifies the minimum frequency of a category or its complement before that category is considered when allcat=TRUE.
iterms	set to TRUE to consider derived terms (dummy variables and nonlinear spline components) as separate variables. This will perform a redundancy analysis on pieces of the variables.
pc	if iterms=TRUE you can set pc to TRUE to replace the submatrix of terms corresponding to each variable with the orthogonal principal components before doing the redundancy analysis. The components are based on the correlation matrix.
pr	set to TRUE to monitor progress of the stepwise algorithm
...	arguments to pass to dataframeReduce to remove "difficult" variables from data if formula is ~. to use all variables in data (data must be specified when these arguments are used). Ignored for print.
x	an object created by redun
digits	number of digits to which to round $R^2$ values when printing
long	set to FALSE to prevent the print method from printing the $R^2$ history and the original $R^2$ with which each variable can be predicted from ALL other variables.

### Details

A categorical variable is deemed redundant if a linear combination of dummy variables representing it can be predicted from a linear combination of other variables. For example, if there were 4 cities in the data and each city's rainfall was also present as a variable, with virtually the same rainfall reported for all observations for a city, city would be redundant given rainfall (or vice-versa; the one declared redundant would be the first one in the formula). If two cities had the same rainfall, city might be declared redundant even though tied cities might be deemed non-redundant in another setting. To ensure that all categories may be predicted well from other variables, use the allcat option. To ignore categories that are too infrequent or too frequent, set minfreq to a nonzero integer. When the number of observations in the category is below this number or the number of observations not in the category is below this number, no attempt is made to predict observations being in that category individually for the purpose of redundancy detection.

### Value

an object of class "redun" including an element "scores", a numeric matrix with all transformed values when each variable was the dependent variable and the first canonical variate was computed

**Author(s)**

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>

**See Also**

[areg](#), [dataframeReduce](#), [transcan](#), [varclus](#), [r2describe](#), [subselect::genetic](#)

**Examples**

```
set.seed(1)
n <- 100
x1 <- runif(n)
x2 <- runif(n)
x3 <- x1 + x2 + runif(n)/10
x4 <- x1 + x2 + x3 + runif(n)/10
x5 <- factor(sample(c('a', 'b', 'c'), n, replace=TRUE))
x6 <- 1*(x5=='a' | x5=='c')
redun(~x1+x2+x3+x4+x5+x6, r2=.8)
redun(~x1+x2+x3+x4+x5+x6, r2=.8, minfreq=40)
redun(~x1+x2+x3+x4+x5+x6, r2=.8, allcat=TRUE)
# x5 is no longer redundant but x6 is
redun(~x1+x2+x3+x4+x5+x6, r2=.8, rank=TRUE)
redun(~x1+x2+x3+x4+x5+x6, r2=.8, qrank=TRUE)

# To help decode which variables made a particular variable redundant:
# r <- redun(...)
# r2describe(r$scores)
```

**Description**

If the first argument is a matrix, reShape strings out its values and creates row and column vectors specifying the row and column each element came from. This is useful for sending matrices to Trellis functions, for analyzing or plotting results of table or crosstabs, or for reformatting serial data stored in a matrix (with rows representing multiple time points) into vectors. The number of observations in the new variables will be the product of the number of rows and number of columns in the input matrix. If the first argument is a vector, the id and colvar variables are used to restructure it into a matrix, with NAs for elements that corresponded to combinations of id and colvar values that did not exist in the data. When more than one vector is given, multiple matrices are created. This is useful for restructuring irregular serial data into regular matrices. It is also useful for converting data produced by expand.grid into a matrix (see the last example). The number of rows of the new matrices equals the number of unique values of id, and the number of columns equals the number of unique values of colvar.

When the first argument is a vector and the `id` is a data frame (even with only one variable), `reShape` will produce a data frame, and the unique groups are identified by combinations of the values of all variables in `id`. If a data frame constant is specified, the variables in this data frame are assumed to be constant within combinations of `id` variables (if not, an arbitrary observation in constant will be selected for each group). A row of constant corresponding to the target `id` combination is then carried along when creating the data frame result.

A different behavior of `reShape` is achieved when `base` and `reps` are specified. In that case `x` must be a list or data frame, and those data are assumed to contain one or more non-repeating measurements (e.g., baseline measurements) and one or more repeated measurements represented by variables named by pasting together the character strings in the vector `base` with the integers 1, 2, ..., `reps`. The input data are rearranged by repeating each value of the baseline variables `reps` times and by transposing each observation's values of one of the set of repeated measurements as `reps` observations under the variable whose name does not have an integer pasted to the end. If `x` has a `row.names` attribute, those observation identifiers are each repeated `reps` times in the output object. See the last example.

### Usage

```
reShape(x, ..., id, colvar, base, reps, times=1:reps,
        timevar='seqno', constant=NULL)
```

### Arguments

<code>x</code>	a matrix or vector, or, when <code>base</code> is specified, a list or data frame
<code>...</code>	other optional vectors, if <code>x</code> is a vector
<code>id</code>	A numeric, character, category, or factor variable containing subject identifiers, or a data frame of such variables that in combination form groups of interest. Required if <code>x</code> is a vector, ignored otherwise.
<code>colvar</code>	A numeric, character, category, or factor variable containing column identifiers. <code>colvar</code> is using a "time of data collection" variable. Required if <code>x</code> is a vector, ignored otherwise.
<code>base</code>	vector of character strings containing base names of repeated measurements
<code>reps</code>	number of times variables named in <code>base</code> are repeated. This must be a constant.
<code>times</code>	when <code>base</code> is given, <code>times</code> is the vector of times to create if you do not want to use consecutive integers beginning with 1.
<code>timevar</code>	specifies the name of the time variable to create if <code>times</code> is given, if you do not want to use <code>seqno</code>
<code>constant</code>	a data frame with the same number of rows in <code>id</code> and <code>x</code> , containing auxiliary information to be merged into the resulting data frame. Logically, the rows of <code>constant</code> within each group should have the same value of all of its variables.

### Details

In converting `dimnames` to vectors, the resulting variables are numeric if all elements of the matrix `dimnames` can be converted to numeric, otherwise the corresponding row or column variable remains character. When the `dimnames` of `x` have a `names` attribute, those two names become the

new variable names. If `x` is a vector and another vector is also given (in `...`), the matrices in the resulting list are named the same as the input vector calling arguments. You can specify customized names for these on-the-fly by using e.g. `reShape(X=x, Y=y, id= , colvar= )`. The new names will then be `X` and `Y` instead of `x` and `y`. A new variable named `seqno` is also added to the resulting object. `seqno` indicates the sequential repeated measurement number. When `base` and `times` are specified, this new variable is named the character value of `timevar` and the values are given by a table lookup into the vector `times`.

### Value

If `x` is a matrix, returns a list containing the row variable, the column variable, and the `as.vector(x)` vector, named the same as the calling argument was called for `x`. If `x` is a vector and no other vectors were specified as `...`, the result is a matrix. If at least one vector was given to `...`, the result is a list containing `k` matrices, where `k` one plus the number of vectors in `...`. If `x` is a list or data frame, the same type of object is returned. If `x` is a vector and `id` is a data frame, a data frame will be the result.

### Author(s)

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University School of Medicine  
 <fh@fharrell.com>

### See Also

[reshape](#), [as.vector](#), [matrix](#), [dimnames](#), [outer](#), [table](#)

### Examples

```
set.seed(1)
Solder <- factor(sample(c('Thin', 'Thick'), 200, TRUE), c('Thin', 'Thick'))
Opening <- factor(sample(c('S', 'M', 'L'), 200, TRUE), c('S', 'M', 'L'))

tab <- table(Opening, Solder)
tab
reShape(tab)
# attach(tab) # do further processing

# An example where a matrix is created from irregular vectors
follow <- data.frame(id=c('a', 'a', 'b', 'b', 'b', 'd'),
                    month=c(1, 2, 1, 2, 3, 2),
                    cholesterol=c(225, 226, 320, 319, 318, 270))

follow
attach(follow)
reShape(cholesterol, id=id, colvar=month)
detach('follow')
# Could have done :
# reShape(cholesterol, triglyceride=trig, id=id, colvar=month)

# Create a data frame, reshaping a long dataset in which groups are
```

```

# formed not just by subject id but by combinations of subject id and
# visit number. Also carry forward a variable that is supposed to be
# constant within subject-visit number combinations. In this example,
# it is not constant, so an arbitrary visit number will be selected.
w <- data.frame(id=c('a','a','a','a','b','b','b','d','d','d'),
               visit=c( 1, 1, 2, 2, 1, 1, 2, 2, 2, 2),
               k=c('A','A','B','B','C','C','D','E','F','G'),
               var=c('x','y','x','y','x','y','y','x','y','z'),
               val=1:10)

with(w,
     reShape(val, id=data.frame(id,visit),
            constant=data.frame(k), colvar=var))

# Get predictions from a regression model for 2 systematically
# varying predictors. Convert the predictions into a matrix, with
# rows corresponding to the predictor having the most values, and
# columns corresponding to the other predictor
# d <- expand.grid(x2=0:1, x1=1:100)
# pred <- predict(fit, d)
# reShape(pred, id=d$x1, colvar=d$x2) # makes 100 x 2 matrix

# Reshape a wide data frame containing multiple variables representing
# repeated measurements (3 repeats on 2 variables; 4 subjects)
set.seed(33)
n <- 4
w <- data.frame(age=rnorm(n, 40, 10),
               sex=sample(c('female','male'), n,TRUE),
               sbp1=rnorm(n, 120, 15),
               sbp2=rnorm(n, 120, 15),
               sbp3=rnorm(n, 120, 15),
               dbp1=rnorm(n, 80, 15),
               dbp2=rnorm(n, 80, 15),
               dbp3=rnorm(n, 80, 15), row.names=letters[1:n])
options(digits=3)
w

u <- reShape(w, base=c('sbp','dbp'), reps=3)
u
reShape(w, base=c('sbp','dbp'), reps=3, timevar='week', times=c(0,3,12))

```

**Description**

rlegend is a version of [legend](#) for R that implements `plot=FALSE`, adds `grid=TRUE`, and defaults `lty`, `lwd`, `pch` to `NULL` and checks for `length>0` rather than `missing()`, so it's easier to deal with non-applicable parameters. But when **grid** is in effect, the preferred function to use is `rlegendg`, which calls the **lattice** `draw.key` function.

**Usage**

```
rlegend(x, y, legend, fill, col = "black", lty = NULL, lwd = NULL,
        pch = NULL, angle = NULL, density = NULL, bty = "o",
        bg = par("bg"), pt.bg = NA, cex = 1, xjust = 0, yjust = 1,
        x.intersp = 1, y.intersp = 1, adj = 0, text.width = NULL,
        merge = do.lines && has.pch, trace = FALSE, ncol = 1,
        horiz = FALSE, plot = TRUE, grid = FALSE, ...)

rlegendg(x, y, legend, col=pr$col[1], lty=NULL,
         lwd=NULL, pch=NULL, cex=pr$cex[1], other=NULL)
```

**Arguments**

x, y, legend, fill, col, lty, lwd, pch, angle, density, bty, bg, pt.bg, cex, xjust, yjust, x.intersp, y.intersp, adj, text.width, merge, trace, ncol, horiz

                                  see [legend](#)

plot                              set to FALSE to suppress drawing the legend. This is used to compute the size needed for when the legend is drawn with a later call to `rlegend`.

grid                              set to TRUE if the **grid** package is in effect

...                               see [legend](#)

other                             a list containing other arguments to pass to `draw.key`. See the help file for [xyplot](#).

**Value**

a list with elements `rect` and `text`. `rect` has elements `w`, `h`, `left`, `top` with size/position information.

**Author(s)**

Frank Harrell and R-Core

**See Also**

[legend](#), [draw.key](#), [xyplot](#)

## Description

For a dataset containing a time variable, a scalar response variable, and an optional subject identification variable, obtains least squares estimates of the coefficients of a restricted cubic spline function or a linear regression in time after adjusting for subject effects through the use of subject dummy variables. Then the fit is bootstrapped  $B$  times, either by treating time and subject ID as fixed (i.e., conditioning the analysis on them) or as random variables. For the former, the residuals from the original model fit are used as the basis of the bootstrap distribution. For the latter, samples are taken jointly from the time, subject ID, and response vectors to obtain unconditional distributions.

If a subject id variable is given, the bootstrap sampling will be based on samples with replacement from subjects rather than from individual data points. In other words, either none or all of a given subject's data will appear in a bootstrap sample. This cluster sampling takes into account any correlation structure that might exist within subjects, so that confidence limits are corrected for within-subject correlation. Assuming that ordinary least squares estimates, which ignore the correlation structure, are consistent (which is almost always true) and efficient (which would not be true for certain correlation structures or for datasets in which the number of observation times vary greatly from subject to subject), the resulting analysis will be a robust, efficient repeated measures analysis for the one-sample problem.

Predicted values of the fitted models are evaluated by default at a grid of 100 equally spaced time points ranging from the minimum to maximum observed time points. Predictions are for the average subject effect. Pointwise confidence intervals are optionally computed separately for each of the points on the time grid. However, simultaneous confidence regions that control the level of confidence for the entire regression curve lying within a band are often more appropriate, as they allow the analyst to draw conclusions about nuances in the mean time response profile that were not stated a priori. The method of *Tibshirani (1997)* is used to easily obtain simultaneous confidence sets for the set of coefficients of the spline or linear regression function as well as the average intercept parameter (over subjects). Here one computes the objective criterion (here both the  $-2$  log likelihood evaluated at the bootstrap estimate of beta but with respect to the original design matrix and response vector, and the sum of squared errors in predicting the original response vector) for the original fit as well as for all of the bootstrap fits. The confidence set of the regression coefficients is the set of all coefficients that are associated with objective function values that are less than or equal to say the 0.95 quantile of the vector of  $B + 1$  objective function values. For the coefficients satisfying this condition, predicted curves are computed at the time grid, and minima and maxima of these curves are computed separately at each time point to derive the final simultaneous confidence band.

By default, the log likelihoods that are computed for obtaining the simultaneous confidence band assume independence within subject. This will cause problems unless such log likelihoods have very high rank correlation with the log likelihood allowing for dependence. To allow for correlation or to estimate the correlation function, see the `cor.pattern` argument below.

## Usage

```
rm.boot(time, y, id=seq(along=time), subset,
        plot.individual=FALSE,
        bootstrap.type=c('x fixed', 'x random'),
        nk=6, knots, B=500, smoother=supsmu,
        xlab, xlim, ylim=range(y),
        times=seq(min(time), max(time), length=100),
```



```

absorb.subject.effects=FALSE,
rho=0, cor.pattern=c('independent','estimate'), ncor=10000,
...)

## S3 method for class 'rm.boot'
plot(x, obj2, conf.int=.95,
      xlab=x$xlabel, ylab=x$ylabel,
      xlim, ylim=x$ylim,
      individual.boot=FALSE,
      pointwise.band=FALSE,
      curves.in.simultaneous.band=FALSE,
      col.pointwise.band=2,
      objective=c('-2 log L','sse','dep -2 log L'), add=FALSE, ncurves,
      multi=FALSE, multi.method=c('color','density'),
      multi.conf =c(.05,.1,.2,.3,.4,.5,.6,.7,.8,.9,.95,.99),
      multi.density=c(-1,90,80,70,60,50,40,30,20,10, 7, 4),
      multi.col =c( 1, 8,20, 5, 2, 7,15,13,10,11, 9, 14),
      subtitles=TRUE, ...)

```

### Arguments

time	numeric time vector
y	continuous numeric response vector of length the same as time. Subjects having multiple measurements have the measurements strung out.
x	an object returned from rm.boot
id	subject ID variable. If omitted, it is assumed that each time-response pair is measured on a different subject.
subset	subset of observations to process if not all the data
plot.individual	set to TRUE to plot nonparametrically smoothed time-response curves for each subject
bootstrap.type	specifies whether to treat the time and subject ID variables as fixed or random
nk	number of knots in the restricted cubic spline function fit. The number of knots may be 0 (denoting linear regression) or an integer greater than 2 in which k knots results in $k - 1$ regression coefficients excluding the intercept. The default is 6 knots.
knots	vector of knot locations. May be specified if nk is omitted.
B	number of bootstrap repetitions. Default is 500.
smoother	a smoothing function that is used if plot.individual=TRUE. Default is <code>supsmu</code> .
xlab	label for x-axis. Default is "units" attribute of the original time variable, or "Time" if no such attribute was defined using the units function.
xlim	specifies x-axis plotting limits. Default is to use range of times specified to rm.boot.

ylim	for <code>rm.boot</code> this is a vector of y-axis limits used if <code>plot.individual=TRUE</code> . It is also passed along for later use by <code>plot.rm.boot</code> . For <code>plot.rm.boot</code> , ylim can be specified, to override the value stored in the object stored by <code>rm.boot</code> . The default is the actual range of y in the input data.
times	a sequence of times at which to evaluate fitted values and confidence limits. Default is 100 equally spaced points in the observed range of time.
absorb.subject.effects	If TRUE, adjusts the response vector y before re-sampling so that the subject-specific effects in the initial model fit are all zero. Then in re-sampling, subject effects are not used in the models. This will downplay one of the sources of variation. This option is used mainly for checking for consistency of results, as the re-sampling analyses are simpler when <code>absorb.subject.effects=TRUE</code> .
rho	The log-likelihood function that is used as the basis of simultaneous confidence bands assumes normality with independence within subject. To check the robustness of this assumption, if rho is not zero, the log-likelihood under multivariate normality within subject, with constant correlation rho between any two time points, is also computed. If the two log-likelihoods have the same ranks across re-samples, allowing the correlation structure does not matter. The agreement in ranks is quantified using the Spearman rank correlation coefficient. The <code>plot</code> method allows the non-zero intra-subject correlation log-likelihood to be used in deriving the simultaneous confidence band. Note that this approach does assume homoscedasticity.
cor.pattern	More generally than using an equal-correlation structure, you can specify a function of two time vectors that generates as many correlations as the length of these vectors. For example, <code>cor.pattern=function(time1,time2) 0.2^(abs(time1-time2)/10)</code> would specify a dampening serial correlation pattern. <code>cor.pattern</code> can also be a list containing vectors x (a vector of absolute time differences) and y (a corresponding vector of correlations). To estimate the correlation function as a function of absolute time differences within subjects, specify <code>cor.pattern="estimate"</code> . The products of all possible pairs of residuals (or at least up to <code>ncor</code> of them) within subjects will be related to the absolute time difference. The correlation function is estimated by computing the sample mean of the products of standardized residuals, stratified by absolute time difference. The correlation for a zero time difference is set to 1 regardless of the <code>lowess</code> estimate. NOTE: This approach fails in the presence of large subject effects; correcting for such effects removes too much of the correlation structure in the residuals.
ncor	the maximum number of pairs of time values used in estimating the correlation function if <code>cor.pattern="estimate"</code>
...	other arguments to pass to smoother if <code>plot.individual=TRUE</code>
obj2	a second object created by <code>rm.boot</code> that can also be passed to <code>plot.rm.boot</code> . This is used for two-sample problems for which the time profiles are allowed to differ between the two groups. The bootstrapped predicted y values for the second fit are subtracted from the fitted values for the first fit so that the predicted mean response for group 1 minus the predicted mean response for group 2 is what is plotted. The confidence bands that are plotted are also for this difference. For the simultaneous confidence band, the objective criterion is taken to be the sum of the objective criteria ( $-2 \log L$ or sum of squared errors) for the separate

	fits for the two groups. The times vectors must have been identical for both calls to <code>rm.boot</code> , although NAs can be inserted by the user of one or both of the time vectors in the <code>rm.boot</code> objects so as to suppress certain sections of the difference curve from being plotted.
<code>conf.int</code>	the confidence level to use in constructing simultaneous, and optionally pointwise, bands. Default is 0.95.
<code>ylab</code>	label for y-axis. Default is the "label" attribute of the original y variable, or "y" if no label was assigned to y (using the <code>label</code> function, for example).
<code>individual.boot</code>	set to TRUE to plot the first 100 bootstrap regression fits
<code>pointwise.band</code>	set to TRUE to draw a pointwise confidence band in addition to the simultaneous band
<code>curves.in.simultaneous.band</code>	set to TRUE to draw all bootstrap regression fits that had a sum of squared errors (obtained by predicting the original y vector from the original time vector and id vector) that was less than or equal to the <code>conf.int</code> quantile of all bootstrapped models (plus the original model). This will show how the point by point max and min were computed to form the simultaneous confidence band.
<code>col.pointwise.band</code>	color for the pointwise confidence band. Default is '2', which defaults to red for default Windows S-PLUS setups.
<code>objective</code>	the default is to use the -2 times log of the Gaussian likelihood for computing the simultaneous confidence region. If neither <code>cor.pattern</code> nor <code>rho</code> was specified to <code>rm.boot</code> , the independent homoscedastic Gaussian likelihood is used. Otherwise the dependent homoscedastic likelihood is used according to the specified or estimated correlation pattern. Specify <code>objective="sse"</code> to instead use the sum of squared errors.
<code>add</code>	set to TRUE to add curves to an existing plot. If you do this, titles and subtitles are omitted.
<code>ncurves</code>	when using <code>individual.boot=TRUE</code> or <code>curves.in.simultaneous.band=TRUE</code> , you can plot a random sample of <code>ncurves</code> of the fitted curves instead of plotting up to B of them.
<code>multi</code>	set to TRUE to draw multiple simultaneous confidence bands shaded with different colors. Confidence levels vary over the values in the <code>multi.conf</code> vector.
<code>multi.method</code>	specifies the method of shading when <code>multi=TRUE</code> . Default is to use colors, with the default colors chosen so that when the graph is printed under S-Plus for Windows 4.0 to an HP LaserJet printer, the confidence regions are naturally ordered by darkness of gray-scale. Regions closer to the point estimates (i.e., the center) are darker. Specify <code>multi.method="density"</code> to instead use densities of lines drawn per inch in the confidence regions, with all regions drawn with the default color. The <code>polygon</code> function is used to shade the regions.
<code>multi.conf</code>	vector of confidence levels, in ascending order. Default is to use 12 confidence levels ranging from 0.05 to 0.99.
<code>multi.density</code>	vector of densities in lines per inch corresponding to <code>multi.conf</code> . As is the convention in the <code>polygon</code> function, a density of -1 indicates a solid region.

`multi.col`        vector of colors corresponding to `multi.conf`. See `multi.method` for rationale.  
`subtitles`        set to FALSE to suppress drawing subtitles for the plot

### Details

Observations having missing time or y are excluded from the analysis.

As most repeated measurement studies consider the times as design points, the fixed covariable case is the default. Bootstrapping the residuals from the initial fit assumes that the model is correctly specified. Even if the covariables are fixed, doing an unconditional bootstrap is still appropriate, and for large sample sizes unconditional confidence intervals are only slightly wider than conditional ones. For moderate to small sample sizes, the `bootstrap.type="x random"` method can be fairly conservative.

If not all subjects have the same number of observations (after deleting observations containing missing values) and if `bootstrap.type="x fixed"`, bootstrapped residual vectors may have a length  $m$  that is different from the number of original observations  $n$ . If  $m > n$  for a bootstrap repetition, the first  $n$  elements of the randomly drawn residuals are used. If  $m < n$ , the residual vector is appended with a random sample with replacement of length  $n - m$  from itself. A warning message is issued if this happens. If the number of time points per subject varies, the bootstrap results for `bootstrap.type="x fixed"` can still be invalid, as this method assumes that a vector (over subjects) of all residuals can be added to the original y-hats, and varying number of points will cause mis-alignment.

For `bootstrap.type="x random"` in the presence of significant subject effects, the analysis is approximate as the subjects used in any one bootstrap fit will not be the entire list of subjects. The average (over subjects used in the bootstrap sample) intercept is used from that bootstrap sample as a predictor of average subject effects in the overall sample.

Once the bootstrap coefficient matrix is stored by `rm.boot`, `plot.rm.boot` can be run multiple times with different options (e.g, different confidence levels).

See `bootcov` in the **rms** library for a general approach to handling repeated measurement data for ordinary linear models, binary and ordinal models, and survival models, using the unconditional bootstrap. `bootcov` does not handle bootstrapping residuals.

### Value

an object of class `rm.boot` is returned by `rm.boot`. The principal object stored in the returned object is a matrix of regression coefficients for the original fit and all of the bootstrap repetitions (object `Coef`), along with vectors of the corresponding  $-2$  log likelihoods are sums of squared errors. The original fit object from `lm.fit.qr` is stored in `fit`. For this fit, a cell means model is used for the id effects.

`plot.rm.boot` returns a list containing the vector of times used for plotting along with the overall fitted values, lower and upper simultaneous confidence limits, and optionally the pointwise confidence limits.

### Author(s)

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University School of Medicine  
 <fh@fharrell.com>

## References

- Feng Z, McLerran D, Grizzle J (1996): A comparison of statistical methods for clustered data analysis with Gaussian error. *Stat in Med* 15:1793–1806.
- Tibshirani R, Knight K (1997): Model search and inference by bootstrap "bumping". Technical Report, Department of Statistics, University of Toronto.  
<https://www.jstor.org/stable/1390820>. Presented at the Joint Statistical Meetings, Chicago, August 1996.
- Efron B, Tibshirani R (1993): *An Introduction to the Bootstrap*. New York: Chapman and Hall.
- Diggle PJ, Verbyla AP (1998): Nonparametric estimation of covariance structure in longitudinal data. *Biometrics* 54:401–415.
- Chapman IM, Hartman ML, et al (1997): Effect of aging on the sensitivity of growth hormone secretion to insulin-like growth factor-I negative feedback. *J Clin Endocrinol Metab* 82:2996–3004.
- Li Y, Wang YG (2008): Smooth bootstrap methods for analysis of longitudinal data. *Stat in Med* 27:937-953. (potential improvements to cluster bootstrap; not implemented here)

## See Also

[rcspline.eval](#), [lm](#), [lowess](#), [supsmu](#), [bootcov](#), [units](#), [label](#), [polygon](#), [reShape](#)

## Examples

```
# Generate multivariate normal responses with equal correlations (.7)
# within subjects and no correlation between subjects
# Simulate realizations from a piecewise linear population time-response
# profile with large subject effects, and fit using a 6-knot spline
# Estimate the correlation structure from the residuals, as a function
# of the absolute time difference

# Function to generate n p-variate normal variates with mean vector u and
# covariance matrix S
# Slight modification of function written by Bill Venables
# See also the built-in function rmvnorm
rmvnorm <- function(n, p = 1, u = rep(0, p), S = diag(p)) {
  Z <- matrix(rnorm(n * p), p, n)
  t(u + t(chol(S)) %*% Z)
}

n <- 20 # Number of subjects
sub <- .5*(1:n) # Subject effects

# Specify functional form for time trend and compute non-stochastic component
times <- seq(0, 1, by=.1)
g <- function(times) 5*pmax(abs(times-.5),.3)
ey <- g(times)
```

```

# Generate multivariate normal errors for 20 subjects at 11 times
# Assume equal correlations of rho=.7, independent subjects

nt  <- length(times)
rho <- .7

set.seed(19)
errors <- mvrnorm(n, p=nt, S=diag(rep(1-rho,nt))+rho)
# Note: first random number seed used gave rise to mean(errors)=0.24!

# Add E[Y], error components, and subject effects
y      <- matrix(rep(ey,n), ncol=nt, byrow=TRUE) + errors +
          matrix(rep(sub,nt), ncol=nt)

# String out data into long vectors for times, responses, and subject ID
y      <- as.vector(t(y))
times  <- rep(times, n)
id     <- sort(rep(1:n, nt))

# Show lowess estimates of time profiles for individual subjects
f <- rm.boot(times, y, id, plot.individual=TRUE, B=25, cor.pattern='estimate',
             smoother=lowess, bootstrap.type='x fixed', nk=6)
# In practice use B=400 or 500
# This will compute a dependent-structure log-likelihood in addition
# to one assuming independence. By default, the dep. structure
# objective will be used by the plot method (could have specified rho=.7)
# NOTE: Estimating the correlation pattern from the residual does not
# work in cases such as this one where there are large subject effects

# Plot fits for a random sample of 10 of the 25 bootstrap fits
plot(f, individual.boot=TRUE, ncurves=10, ylim=c(6,8.5))

# Plot pointwise and simultaneous confidence regions
plot(f, pointwise.band=TRUE, col.pointwise=1, ylim=c(6,8.5))

# Plot population response curve at average subject effect
ts <- seq(0, 1, length=100)
lines(ts, g(ts)+mean(sub), lwd=3)

## Not run:
#
# Handle a 2-sample problem in which curves are fitted
# separately for males and females and we wish to estimate the

```

```

# difference in the time-response curves for the two sexes.
# The objective criterion will be taken by plot.rm.boot as the
# total of the two sums of squared errors for the two models
#
knots <- rcspline.eval(c(time.f,time.m), nk=6, knots.only=TRUE)
# Use same knots for both sexes, and use a times vector that
# uses a range of times that is included in the measurement
# times for both sexes
#
tm <- seq(max(min(time.f),min(time.m)),
          min(max(time.f),max(time.m)),length=100)

f.female <- rm.boot(time.f, bp.f, id.f, knots=knots, times=tm)
f.male  <- rm.boot(time.m, bp.m, id.m, knots=knots, times=tm)
plot(f.female)
plot(f.male)
# The following plots female minus male response, with
# a sequence of shaded confidence band for the difference
plot(f.female,f.male,multi=TRUE)

# Do 1000 simulated analyses to check simultaneous coverage
# probability. Use a null regression model with Gaussian errors

n.per.pt <- 30
n.pt      <- 10

null.in.region <- 0

for(i in 1:1000) {
  y      <- rnorm(n.pt*n.per.pt)
  time   <- rep(1:n.per.pt, n.pt)
  # Add the following line and add ,id=id to rm.boot to use clustering
  # id    <- sort(rep(1:n.pt, n.per.pt))
  # Because we are ignoring patient id, this simulation is effectively
  # using 1 point from each of 300 patients, with times 1,2,3,,30

  f <- rm.boot(time, y, B=500, nk=5, bootstrap.type='x fixed')
  g <- plot(f, ylim=c(-1,1), pointwise=FALSE)
  null.in.region <- null.in.region + all(g$lower<=0 & g$upper>=0)
  prn(c(i=i,null.in.region=null.in.region))
}

# Simulation Results: 905/1000 simultaneous confidence bands
# fully contained the horizontal line at zero

## End(Not run)

```

---

`rMultinom`*Generate Multinomial Random Variables with Varying Probabilities*

---

**Description**

Given a matrix of multinomial probabilities where rows correspond to observations and columns to categories (and each row sums to 1), generates a matrix with the same number of rows as has probs and with `m` columns. The columns represent multinomial cell numbers, and within a row the columns are all samples from the same multinomial distribution. The code is a modification of that in the `impute.polyreg` function in the MICE package.

**Usage**

```
rMultinom(probs, m)
```

**Arguments**

<code>probs</code>	matrix of probabilities
<code>m</code>	number of samples for each row of probs

**Value**

an integer matrix having `m` columns

**See Also**

[rbinom](#)

**Examples**

```
set.seed(1)
w <- rMultinom(rbind(c(.1,.2,.3,.4),c(.4,.3,.2,.1)),200)
t(apply(w, 1, table)/200)
```

---

`runifChanged`*runifChanged*

---

**Description**

Re-run Code if an Input Changed

**Usage**

```
runifChanged(fun, ..., file = NULL, .print. = TRUE, .inclfun. = TRUE)
```



**Arguments**

<code>fun</code>	the (usually slow) function to run
<code>...</code>	input objects the result of running the function is dependent on
<code>file</code>	file in which to store the result of <code>fun</code> augmented by attributes containing hash digests
<code>.print.</code>	set to <code>TRUE</code> to list which objects changed that necessitated re-running <code>f</code>
<code>.inclfun.</code>	set to <code>FALSE</code> to not include <code>fun</code> in the hash digest, i.e., to not require re-running <code>fun</code> if only <code>fun</code> itself has changed

**Details**

Uses `hashCheck` to run a function and save the results if specified inputs have changed, otherwise retrieve results from a file. This makes it easy to see if any objects changed that require re-running a long simulation, and reports on any changes. The file name is taken as the chunk name appended with `.rds` unless it is given as `file=`. `fun` has no arguments. Set `.inclfun.=FALSE` to not include `fun` in the hash check (for legacy uses). The typical workflow is as follows.

```
f <- function(      ) {
# . . . do the real work with multiple function calls ...
}
seed <- 3
set.seed(seed)
w <- runifChanged(f, seed, obj1, obj2, ....)
```

`seed`, `obj1`, `obj2`, ... are all the objects that `f()` uses that if changed would give a different result of `f()`. This can include functions such as those in a package, and `f` will be re-run if any of the function's code changes. `f` is also re-run if the code inside `f` changes. The result of `f` is stored with `saveRDS` by default in file named `xxx.rds` where `xxx` is the label for the current chunk. To control this use instead `file=xxx.rds` add the file argument to `runifChanged(...)`. If nothing has changed and the file already exists, the file is read to create the result object (e.g., `w` above). If `f()` needs to be run, the hashed input objects are stored as attributes for the result then the enhanced result is written to the file.

See [here](#) for examples.

**Value**

the result of running `fun`

**Author(s)**

Frank Harrell

runParallel

*runParallel***Description**

parallel Package Easy Front-End

**Usage**

```
runParallel(
  onecore,
  reps,
  seed = round(runif(1, 0, 10000)),
  cores = max(1, parallel::detectCores() - 1),
  simplify = TRUE,
  along
)
```

**Arguments**

onecore	function to run the analysis on one core
reps	total number of repetitions
seed	species the base random number seed. The seed used for core <i>i</i> will be seed + <i>i</i> .
cores	number of cores to use, defaulting to one less than the number available
simplify	set to FALSE to not create an outer list if a onecore result has only one element
along	see Details

**Details**

Given a function `onecore` that runs the needed set of simulations on one CPU core, and given a total number of repetitions `reps`, determines the number of available cores and by default uses one less than that. By default the number of cores is one less than the number available on your machine. `reps` is divided as evenly as possible over these cores, and batches are run on the cores using the `parallel` package `mclapply` function. The current per-core repetition number is continually updated in your system's temporary directory (`/tmp` for Linux and Mac, `TEMP` for Windows) in a file name `progressX.log` where `X` is the core number. The random number seed is set for each core and is equal to the scalar `seed` - core number + 1. The default seed is a random number between 0 and 10000 but it's best if the user provides the seed so the simulation is reproducible. The total run time is computed and printed. `onefile` must create a named list of all the results created during that one simulation batch. Elements of this list must be data frames, vectors, matrices, or arrays. Upon completion of all batches, all the results are `rbind`'d and saved in a single list.

`onecore` must have an argument `reps` that will tell the function how many simulations to run for one batch, another argument `showprogress` which is a function to be called inside `onecore` to write to the progress file for the current core and repetition, and an argument `core` which informs `onecore`

which sequential core number (batch number) it is processing. When calling `showprogress` inside `onecore`, the arguments, in order, must be the integer value of the repetition to be noted, the number of reps, core, an optional 4th argument `other` that can contain a single character string to add to the output, and an optional 5th argument `pr`. You can set `pr=FALSE` to suppress printing and have `showprogress` return the file name for holding progress information if you want to customize printing.

If any of the objects appearing as list elements produced by `onecore` are multi-dimensional arrays, you must specify an integer value for `along`. This specifies to the `abind` package `abind` function the dimension along which to bind the arrays. For example, if the first dimension of the array corresponding to repetitions, you would specify `along=1`. All arrays present must use the same `along` unless `along` is a named vector and the names match elements of the simulation result object. Set `simplify=FALSE` if you don't want the result simplified if `onecore` produces only one list element. The default returns the first (and only) list element rather than the list if there is only one element.

See [here](#) for examples.

### Value

result from combining all the parallel runs, formatting as similar to the result produced from one run as possible

### Author(s)

Frank Harrell

---

samplesize.bin

*Sample Size for 2-sample Binomial*

---

### Description

Computes sample size(s) for 2-sample binomial problem given vector or scalar probabilities in the two groups.

### Usage

```
samplesize.bin(alpha, beta, pit, pic, rho=0.5)
```

### Arguments

<code>alpha</code>	scalar ONE-SIDED test size, or two-sided size/2
<code>beta</code>	scalar or vector of powers
<code>pit</code>	hypothesized treatment probability of success
<code>pic</code>	hypothesized control probability of success
<code>rho</code>	proportion of the sample devoted to treated group ( $0 < \text{rho} < 1$ )

### Value

TOTAL sample size(s)

**AUTHOR**

Rick Chappell  
 Dept. of Statistics and Human Oncology  
 University of Wisconsin at Madison  
 <chappell@stat.wisc.edu>

**Examples**

```
alpha <- .05
beta <- c(.70,.80,.90,.95)

# N1 is a matrix of total sample sizes whose
# rows vary by hypothesized treatment success probability and
# columns vary by power
# See Meinert's book for formulae.

N1 <- samplesize.bin(alpha, beta, pit=.55, pic=.5)
N1 <- rbind(N1, samplesize.bin(alpha, beta, pit=.60, pic=.5))
N1 <- rbind(N1, samplesize.bin(alpha, beta, pit=.65, pic=.5))
N1 <- rbind(N1, samplesize.bin(alpha, beta, pit=.70, pic=.5))
attr(N1,"dimnames") <- NULL

#Accounting for 5% noncompliance in the treated group
inflation <- (1/.95)**2
print(round(N1*inflation+.5,0))
```

---

sas.get

---

*Convert a SAS Dataset to an S Data Frame*


---

**Description**

Converts a SAS dataset into an S data frame. You may choose to extract only a subset of variables or a subset of observations in the SAS dataset. You may have the function automatically convert

PROC FORMAT

-coded variables to factor objects. The original SAS codes are stored in an attribute called `sas.codes` and these may be added back to the levels of a factor variable using the `code.levels` function. Information about special missing values may be captured in an attribute of each variable having special missing values. This attribute is called `special.miss`, and such variables are given class `special.miss`. There are `print`, `[]`, `format`, and `is.special.miss` methods for such variables. The `chron` function is used to set up date, time, and date-time variables. If using S-Plus 5 or 6 or later, the `timeDate` function is used instead. Under R, [Dates](#) is used for dates and [chron](#) for date-times. For times without dates, these still need to be stored in date-time format in POSIX. Such SAS time variables are given a major class of `POSIXt` and a `format.POSIXt` function so that the

date portion (which will always be 1/1/1970) will not print by default. If a date variable represents a partial date (0.5 added if month missing, 0.25 added if day missing, 0.75 if both), an attribute `partial.date` is added to the variable, and the variable also becomes a class imputed variable. The `describe` function uses information about partial dates and special missing values. There is an option to automatically uncompress (or `gunzip`) compressed SAS datasets.

## Usage

```
sas.get(libraryName, member, variables=character(0), ifs=character(0),
        format.library=libraryName, id,
        dates.=c("sas", "yymmdd", "yearfrac", "yearfrac2"),
        keep.log=TRUE, log.file="_temp_.log", macro=sas.get.macro,
        data.frame.out=existsFunction("data.frame"), clean.up=FALSE, quiet=FALSE,
        temp=tempfile("SaS"), formats=TRUE, recode=formats,
        special.miss=FALSE, sasprog="sas",
        as.is=.5, check.unique.id=TRUE, force.single=FALSE,
        pos, uncompress=FALSE, defaultencoding="latin1", var.case="lower")
```

```
is.special.miss(x, code)
```

```
## S3 method for class 'special.miss'
x[... , drop=FALSE]
```

```
## S3 method for class 'special.miss'
print(x, ...)
```

```
## S3 method for class 'special.miss'
format(x, ...)
```

```
sas.codes(object)
```

```
code.levels(object)
```

## Arguments

<code>libraryName</code>	character string naming the directory in which the dataset is kept.
<code>drop</code>	logical. If TRUE the result is coerced to the lowest possible dimension.
<code>member</code>	character string giving the second part of the two part SAS dataset name. (The first part is irrelevant here - it is mapped to the UNIX directory name.)
<code>x</code>	a variable that may have been created by <code>sas.get</code> with <code>special.miss=T</code> or with <code>recode</code> in effect.
<code>variables</code>	vector of character strings naming the variables in the SAS dataset. The S dataset will contain only those variables from the SAS dataset. To get all of the variables (the default), an empty string may be given. It is a fatal error if any one of the variables is not in the SAS dataset. You can use <code>sas.contents</code> to get the variables in the SAS dataset. If you have retrieved a subset of the variables in the SAS dataset and which to retrieve the same list of variables from another dataset, you can program the value of <code>variables</code> - see one of the last examples.

<code>ifs</code>	a vector of character strings, each containing one SAS “subsetting if” statement. These will be used to extract a subset of the observations in the SAS dataset.
<code>format.library</code>	The UNIX directory containing the file ‘ <code>formats.sct</code> ’, which contains the definitions of the user defined formats used in this dataset. By default, we look for the formats in the same directory as the data. The user defined formats must be available (so SAS can read the data).
<code>formats</code>	Set <code>formats</code> to <code>FALSE</code> to keep <code>sas.get</code> from telling the SAS macro to retrieve value label formats from <code>format.library</code> . When you do not specify <code>formats</code> or <code>recode</code> , <code>sas.get</code> will set <code>format</code> to <code>TRUE</code> if a SAS format catalog (‘ <code>.sct</code> ’ or ‘ <code>.sc2</code> ’) file exists in <code>format.library</code> . Value label formats if present are stored as the <code>formats</code> attribute of the returned object (see below). A format is used if it is referred to by one or more variables in the dataset, if it contains no ranges of values (i.e., it identifies value labels for single values), and if it is a character format or a numeric format that is not used just to label missing values. If you set <code>recode</code> to <code>TRUE</code> , <code>1</code> , or <code>2</code> , <code>formats</code> defaults to <code>TRUE</code> . To fetch the values and labels for variable <code>x</code> in the dataset <code>d</code> you could type: <pre>f &lt;- attr(d\[x, "format") formats &lt;- attr(d, "formats") formats\[f]\\$values; formats\[f]\\$labels</pre>
<code>recode</code>	This parameter defaults to <code>TRUE</code> if <code>formats</code> is <code>TRUE</code> . If it is <code>TRUE</code> , variables that have an appropriate format (see above) are recoded as factor objects, which map the values to the value labels for the format. Alternatively, set <code>recode</code> to <code>1</code> to use labels of the form <code>value:label</code> , e.g. <code>1:good 2:better 3:best</code> . Set <code>recode</code> to <code>2</code> to use labels such as <code>good(1) better(2) best(3)</code> . Since <code>sas.codes</code> and <code>code.levels</code> add flexibility, the usual choice for <code>recode</code> is <code>TRUE</code> .
<code>special.miss</code>	For numeric variables, any missing values are stored as <code>NA</code> in <code>S</code> . You can recover special missing values by setting <code>special.miss</code> to <code>TRUE</code> . This will cause the <code>special.miss</code> attribute and the <code>special.miss</code> class to be added to each variable that has at least one special missing value. Suppose that variable <code>y</code> was <code>.E</code> in observation 3 and <code>.G</code> in observation 544. The <code>special.miss</code> attribute for <code>y</code> then has the value <pre>list(codes=c("E", "G"), obs=c(3, 544))</pre> To fetch this information for variable <code>y</code> you would say for example <pre>s &lt;- attr(y, "special.miss") s\[codes]; s\[obs]</pre> or use <code>is.special.miss(x)</code> or the <code>print.special.miss</code> method, which will replace <code>NA</code> values for the variable with ‘ <code>E</code> ’ or ‘ <code>G</code> ’ if they correspond to special missing values. The <code>describe</code> function uses this information in printing a data summary.
<code>id</code>	The name of the variable to be used as the row names of the <code>S</code> dataset. The <code>id</code> variable becomes the <code>row.names</code> attribute of a data frame, but the <code>id</code> variable is still retained as a variable in the data frame. (if <code>data.frame.out</code> is <code>FALSE</code> , this will be the attribute ‘ <code>id</code> ’ of the <code>R</code> dataset.) You can also specify a vector of variable names as the <code>id</code> parameter. After fetching the data from SAS, all these variables will be converted to character format and concatenated (with a space as a separator) to form a (hopefully) unique identification variable.
<code>dates.</code>	specifies the format for storing SAS dates in the resulting data frame

as.is	<p>IF <code>data.frame.out = TRUE</code>, SAS character variables are converted to S factor objects if <code>as.is = FALSE</code> or if <code>as.is</code> is a number between 0 and 1 inclusive and the number of unique values of the variable is less than the number of observations (<math>n</math>) times <code>as.is</code>. The default if <code>as.is</code> is 0.5, so character variables are converted to factors only if they have fewer than <math>n/2</math> unique values. The primary purpose of this is to keep unique identification variables as character values in the data frame instead of using more space to store both the integer factor codes and the factor labels.</p>
check.unique.id	<p>If <code>id</code> is specified, the row names are checked for uniqueness if <code>check.unique.id = TRUE</code>. If any are duplicated, a warning is printed. Note that if a data frame is being created with duplicate row names, statements such as <code>my.data.frame["B23", ]</code> will retrieve only the first row with a row name of</p> <p>B23</p> <p>.</p>
force.single	<p>By default, SAS numeric variables having <code>LENGTH &gt; 4</code> are stored as S double precision numerics, which allow for the same precision as a SAS</p> <p>LENGTH</p> <p>8 variable. Set <code>force.single = TRUE</code> to store every numeric variable in single precision (7 digits of precision). This option is useful when the creator of the SAS dataset has failed to use a</p> <p>LENGTH</p> <p>statement. R does not have single precision, so no attempt is made to convert to single if running R.</p>
dates	<p>One of the character strings "sas", "yearfrac", "yearfrac2", "yymmdd". If a SAS variable has a date format (one of "DATE", "MMDDYY", "YYMMDD", "DDMMYY", "YYQ", "MONYY", "JULIAN"), it will be converted to the format specified by <code>dates</code> before being given to S. "sas" gives days from 1/1/1960 (from 1/1/1970 if using <code>chron</code>), "yearfrac" gives days from 1/1/1900 divided by 365.25, "yearfrac2" gives year plus fraction of current year, and "yymmdd" gives a 6 digit number</p> <p>YYMMDD</p> <p>(<code>year%%100</code>, month, day). Note that R will store these as numbers, not as character strings. If <code>dates="sas"</code> and a variable has one of the SAS date formats listed above, the variable will be given a class of 'date' to work with Terry Therneau's implementation of the 'date' class in S. If the <code>chron</code> package or <code>timeDate</code> function is available, these are used instead.</p>
keep.log	logical flag: if FALSE, delete the SAS log file upon completion.
log.file	the name of the SAS log file.
macro	the name of an S object in the current search path that contains the text of the SAS macro called by R. The R object is a character vector that can be edited using for example <code>sas.get.macro &lt;- editor(sas.get.macro)</code> .
data.frame.out	logical flag: if TRUE, the return value will be an S data frame, otherwise it will be a list.

<code>clean.up</code>	logical flag: if TRUE, remove all temporary files when finished. You may want to keep these while debugging the SAS macro. Not needed for R.
<code>quiet</code>	logical flag: if FALSE, print the contents of the SAS log file if there has been an error.
<code>temp</code>	the prefix to use for the temporary files. Two characters will be added to this, the resulting name must fit on your file system.
<code>sasprog</code>	the name of the system command to invoke SAS
<code>uncompress</code>	set to TRUE to automatically invoke the UNIX gunzip command (if <code>'member.ssd01.gz'</code> exists) or the <code>uncompress</code> command (if <code>'member.ssd01.Z'</code> exists) to uncompress the SAS dataset before proceeding. This assumes you have the file permissions to allow uncompressing in place. If the file is already uncompressed, this option is ignored.
<code>pos</code>	by default, a list or data frame which contains all the variables is returned. If you specify <code>pos</code> , each individual variable is placed into a separate object (whose name is the name of the variable) using the <code>assign</code> function with the <code>pos</code> argument. For example, you can put each variable in its own file in a directory, which in some cases may save memory over attaching a data frame.
<code>code</code>	a special missing value code ('A' through 'Z' or '\_') to check against. If <code>code</code> is omitted, <code>is.special.miss</code> will return a TRUE for each observation that has any special missing value.
<code>defaultencoding</code>	encoding to assume if the SAS dataset does not specify one. Defaults to "latin1".
<code>var.case</code>	default is to change case of SAS variable names to lower case. Specify alternatively "upper" or "preserve".
<code>object</code>	a variable in a data frame created by <code>sas.get</code>
<code>...</code>	ignored

## Details

If you specify `special.miss = TRUE` and there are no special missing values in the data SAS dataset, the SAS step will bomb.

For variables having a

PROC FORMAT VALUE

format with some of the levels undefined, `sas.get` will interpret those values as NA if you are using `recode`.

The SAS macro `'sas_get'` uses record lengths of up to 4096 in two places. If you are exporting records that are very long (because of a large number of variables and/or long character variables), you may want to edit these

LRECL

s to quadruple them, for example.



**Value**

if `data.frame.out` is TRUE, the output will be a data frame resembling the SAS dataset. If `id` was specified, that column of the data frame will be used as the row names of the data frame. Each variable in the data frame or vector in the list will have the attributes `label` and `format` containing SAS labels and formats. Underscores in formats are converted to periods. Formats for character variables have `\$` placed in front of their names. If `formats` is TRUE and there are any appropriate format definitions in `format.library`, the returned object will have attribute `formats` containing lists named the same as the format names (with periods substituted for underscores and character formats prefixed by `\$`). Each of these lists has a vector called `values` and one called `labels` with the

```
PROC FORMAT; VALUE ...
```

definitions.

If `data.frame.out` is FALSE, the output will be a list of vectors, each containing a variable from the SAS dataset. If `id` was specified, that element of the list will be used as the `id` attribute of the entire list.

**Side Effects**

if a SAS error occurs and `quiet` is FALSE, then the SAS log file will be printed under the control of the less pager.

**BACKGROUND**

The references cited below explain the structure of SAS datasets and how they are stored under UNIX. See SAS *Language* for a discussion of the “subsetting if” statement.

**Note**

You must be able to run SAS (by typing `sas`) on your system. If the S command `!sas` does not start SAS, then this function cannot work.

If you are reading time or date-time variables, you will need to execute the command `library(chron)` to print those variables or the data frame if the `timeDate` function is not available.

**Author(s)**

Terry Therneau, Mayo Clinic  
Frank Harrell, Vanderbilt University  
Bill Dunlap, University of Washington and Insightful Corporation  
Michael W. Kattan, Cleveland Clinic Foundation  
Reinhold Koch (encoding)

**References**

SAS Institute Inc. (1990). *SAS Language: Reference, Version 6*. First Edition. SAS Institute Inc., Cary, North Carolina.

SAS Institute Inc. (1988). SAS Technical Report P-176, *Using the SAS System, Release 6.03, under UNIX Operating Systems and Derivatives*. SAS Institute Inc., Cary, North Carolina.

SAS Institute Inc. (1985). *SAS Introductory Guide*. Third Edition. SAS Institute Inc., Cary, North Carolina.

### See Also

[data.frame](#), [describe](#), [label](#), [upData](#), [cleanup.import](#)

### Examples

```
## Not run:
sas.contents("saslib", "mice")
# [1] "dose" "ld50" "strain" "lab_no"
attr(,"n"):
# [1] 117
mice <- sas.get("saslib", mem="mice", var=c("dose", "strain", "ld50"))
plot(mice$dose, mice$ld50)

nude.mice <- sas.get(lib=unix("echo $HOME/saslib"), mem="mice",
ifs="if strain='nude'")

nude.mice.dl <- sas.get(lib=unix("echo $HOME/saslib"), mem="mice",
var=c("dose", "ld50"), ifs="if strain='nude'")

# Get a dataset from current directory, recode PROC FORMAT; VALUE \dots
# variables into factors with labels of the form "good(1)" "better(2)",
# get special missing values, recode missing codes .D and .R into new
# factor levels "Don't know" and "Refused to answer" for variable q1
d <- sas.get(".", "mydata", recode=2, special.miss=TRUE)
attach(d)
n1 <- length(levels(q1))
lev <- c(levels(q1), "Don't know", "Refused")
q1.new <- as.integer(q1)
q1.new[is.special.miss(q1,"D")] <- n1+1
q1.new[is.special.miss(q1,"R")] <- n1+2
q1.new <- factor(q1.new, 1:(n1+2), lev)
# Note: would like to use factor() in place of as.integer \dots but
# factor in this case adds "NA" as a category level

d <- sas.get(".", "mydata")
sas.codes(d$x) # for PROC FORMATted variables returns original data codes
d$x <- code.levels(d$x) # or attach(d); x <- code.levels(x)
# This makes levels such as "good" "better" "best" into e.g.
# "1:good" "2:better" "3:best", if the original SAS values were 1,2,3

# Retrieve the same variables from another dataset (or an update of
```

```

# the original dataset)
mydata2 <- sas.get('mydata2', var=names(d))
# This only works if none of the original SAS variable names contained _
mydata2 <- cleanup.import(mydata2) # will make true integer variables

# Code from Don MacQueen to generate SAS dataset to test import of
# date, time, date-time variables
# data ssd.test;
#   d1='3mar2002'd ;
#   dt1='3mar2002 9:31:02'dt;
#   t1='11:13:45't;
#   output;
#
#   d1='3jun2002'd ;
#   dt1='3jun2002 9:42:07'dt;
#   t1='11:14:13't;
#   output;
#   format d1 mmdyy10. dt1 datetime. t1 time.;
# run;

## End(Not run)

```

---

sasxport.get

*Enhanced Importing of SAS Transport Files using read.xport*


---

## Description

Uses the `read.xport` and `lookup.xport` functions in the `foreign` library to import SAS datasets. SAS date, time, and date/time variables are converted respectively to `Date`, `POSIX`, or `POSIXct` objects in `R`, variable names are converted to lower case, SAS labels are associated with variables, and (by default) integer-valued variables are converted from storage mode `double` to `integer`. If the user ran `PROC FORMAT CNTLOUT=` in SAS and included the resulting dataset in the SAS version 5 transport file, variables having customized formats that do not include any ranges (i.e., variables having standard `PROC FORMAT; VALUE` label formats) will have their format labels looked up, and these variables are converted to `S` factors.

For those users having access to SAS, `method='csv'` is preferred when importing several SAS datasets. Run SAS macro `exportlib.sas` available from <https://github.com/harrelfe/Hmisc/blob/master/src/sas/exportlib.sas> to convert all SAS datasets in a SAS data library (from any engine supported by your system) into CSV files. If any customized formats are used, it is assumed that the `PROC FORMAT CNTLOUT=` dataset is in the data library as a regular SAS dataset, as above.

`SASdsLabels` reads a file containing `PROC CONTENTS` printed output to parse dataset labels, assuming that `PROC CONTENTS` was run on an entire library.

## Usage

```

sasxport.get(file, lowernames=TRUE, force.single = TRUE,
             method=c('read.xport','dataload','csv'), formats=NULL, allow=NULL,

```

```

out=NULL, keep=NULL, drop=NULL, as.is=0.5, FUN=NULL)
sasdsLabels(file)

```

### Arguments

file	name of a file containing the SAS transport file. file may be a URL beginning with <code>https://</code> . For <code>sasdsLabels</code> , file is the name of a file containing a PROC CONTENTS output listing. For <code>method='csv'</code> , file is the name of the directory containing all the CSV files created by running the <code>exportlib</code> SAS macro.
lowernames	set to FALSE to keep from converting SAS variable names to lower case
force.single	set to FALSE to keep integer-valued variables not exceeding $2^{31} - 1$ in value from being converted to integer storage mode
method	set to "dataload" if you have the dataload executable installed and want to use it instead of <code>read.xport</code> . This seems to correct some errors in which rarely some factor variables are always missing when read by <code>read.xport</code> when in fact they have some non-missing values.
formats	a data frame or list (like that created by <code>read.xport</code> ) containing PROC FORMAT output, if such output is not stored in the main transport file.
allow	a vector of characters allowed by R that should not be converted to periods in variable names. By default, underscores in variable names are converted to periods as with R before version 1.9.
out	a character string specifying a directory in which to write separate R save files (.rda files) for each regular dataset. Each file and the data frame inside it is named with the SAS dataset name translated to lower case and with underscores changed to periods. The default NULL value of out results in a data frame or a list of data frames being returned. When out is given, <code>sasxport.get</code> returns only metadata (see below), invisibly. out only works with <code>methods='csv'</code> . out should not have a trailing slash.
keep	a vector of names of SAS datasets to process (original SAS upper case names). Must include PROC FORMAT dataset if it exists, and if the kept datasets use any of its value label formats.
drop	a vector of names of SAS datasets to ignore (original SAS upper case names)
as.is	SAS character variables are converted to S factor objects if <code>as.is=FALSE</code> or if <code>as.is</code> is a number between 0 and 1 inclusive and the number of unique values of the variable is less than the number of observations (n) times <code>as.is</code> . The default if <code>as.is</code> is .5, so character variables are converted to factors only if they have fewer than $n/2$ unique values. The primary purpose of this is to keep unique identification variables as character values in the data frame instead of using more space to store both the integer factor codes and the factor labels.
FUN	an optional function that will be run on each data frame created, when <code>method='csv'</code> and <code>out</code> are specified. The result of all the FUN calls is made into a list corresponding to the SAS datasets that are read. This list is the FUN attribute of the result returned by <code>sasxport.get</code> .

**Details**

See [contents.list](#) for a way to print the directory of SAS datasets when more than one was imported.

**Value**

If there is more than one dataset in the transport file other than the PROC FORMAT file, the result is a list of data frames containing all the non-PROC FORMAT datasets. Otherwise the result is the single data frame. There is an exception if `out` is specified; that causes separate R save files to be written and the returned value to be a list corresponding to the SAS datasets, with key PROC CONTENTS information in a data frame making up each part of the list. `sasdsLabels` returns a named vector of dataset labels, with names equal to the dataset names.

**Author(s)**

Frank E Harrell Jr

**See Also**

[read.xport](#), [label](#), [sas.get](#), [Dates](#), [DateTimeClasses](#), [lookup.xport](#), [contents](#), [describe](#)

**Examples**

```
## Not run:
# SAS code to generate test dataset:
# libname y SASV5XPT "test2.xpt";
#
# PROC FORMAT; VALUE race 1=green 2=blue 3=purple; RUN;
# PROC FORMAT CNTLOUT=format;RUN; * Name, e.g. 'format', unimportant;
# data test;
# LENGTH race 3 age 4;
# age=30; label age="Age at Beginning of Study";
# race=2;
# d1='3mar2002'd ;
# dt1='3mar2002 9:31:02'dt;
# t1='11:13:45't;
# output;
#
# age=31;
# race=4;
# d1='3jun2002'd ;
# dt1='3jun2002 9:42:07'dt;
# t1='11:14:13't;
# output;
# format d1 mmdyy10. dt1 datetime. t1 time. race race.;
# run;
# data z; LENGTH x3 3 x4 4 x5 5 x6 6 x7 7 x8 8;
# DO i=1 TO 100;
#     x3=ranuni(3);
#     x4=ranuni(5);
#     x5=ranuni(7);
```

```

#       x6=ranuni(9);
#       x7=ranuni(11);
#       x8=ranuni(13);
#       output;
#       END;
#   DROP i;
#   RUN;
# PROC MEANS; RUN;
# PROC COPY IN=work OUT=y;SELECT test format z;RUN; *Creates test2.xpt;
w <- sasxport.get('test2.xpt')
# To use an existing copy of test2.xpt available on the web:
w <- sasxport.get('https://github.com/harrelfe/Hmisc/raw/master/inst/tests/test2.xpt')

describe(w$test) # see labels, format names for dataset test
# Note: if only one dataset (other than format) had been exported,
# just do describe(w) as sasxport.get would not create a list for that
lapply(w, describe)# see descriptive stats for both datasets
contents(w$test) # another way to see variable attributes
lapply(w, contents)# show contents of both datasets
options(digits=7) # compare the following matrix with PROC MEANS output
t(sapply(w$z, function(x)
  c(Mean=mean(x),SD=sqrt(var(x)),Min=min(x),Max=max(x))))

## End(Not run)

```

---

Save

*Faciliate Use of save and load to Remote Directories*


---

## Description

These functions are slightly enhanced versions of `save` and `load` that allow a target directory to be specified using `options(LoadPath="pathname")`. If the `LoadPath` option is not set, the current working directory is used.

## Usage

```

# options(LoadPath='mypath')
Save(object, name=deparse(substitute(object)), compress=TRUE)
Load(object)

```

## Arguments

<code>object</code>	the name of an object, usually a data frame. It must not be quoted.
<code>name</code>	an optional name to assign to the object and file name prefix, if the argument name is not used
<code>compress</code>	see <a href="#">save</a> . Default is TRUE which corresponds to gzip.

## Details

Save creates a temporary version of the object under the name given by the user, so that save will internalize this name. Then subsequent Load or load will cause an object of the original name to be created in the global environment. The name of the R data file is assumed to be the name of the object (or the value of name) appended with ".rda".

## Author(s)

Frank Harrell

## See Also

[save](#), [load](#)

## Examples

```
## Not run:
d <- data.frame(x=1:3, y=11:13)
options(LoadPath='../data/rda')
Save(d) # creates ../data/rda/d.rda
Load(d) # reads ../data/rda/d.rda
Save(d, 'D') # creates object D and saves it in ../D.rda

## End(Not run)
```

---

scat1d

*One-Dimensional Scatter Diagram, Spike Histogram, or Density*

---

## Description

scat1d adds tick marks (bar codes. rug plot) on any of the four sides of an existing plot, corresponding with non-missing values of a vector  $x$ . This is used to show the data density. Can also place the tick marks along a curve by specifying  $y$ -coordinates to go along with the  $x$  values.

If any two values of  $x$  are within  $\text{eps} * w$  of each other, where  $\text{eps}$  defaults to .001 and  $w$  is the span of the intended axis, values of  $x$  are jittered by adding a value uniformly distributed in  $[-\text{jitfrac} * w, \text{jitfrac} * w]$ , where  $\text{jitfrac}$  defaults to .008. Specifying  $\text{preserve}=\text{TRUE}$  invokes `jitter2` with a different logic of jittering. Allows plotting random sub-segments to handle very large  $x$  vectors (`seetfrac`).

`jitter2` is a generic method for jittering, which does not add random noise. It retains unique values and ranks, and randomly spreads duplicate values at equidistant positions within limits of enclosing values. `jitter2` is especially useful for numeric variables with discrete values, like rating scales. Missing values are allowed and are returned. Currently implemented methods are `jitter2.default` for vectors and `jitter2.data.frame` which returns a data.frame with each numeric column jittered.

`datadensity` is a generic method used to show data densities in more complex situations. Here, another `datadensity` method is defined for data frames. Depending on the which argument, some

or all of the variables in a data frame will be displayed, with `scat1d` used to display continuous variables and, by default, bars used to display frequencies of categorical, character, or discrete numeric variables. For such variables, when the total length of value labels exceeds 200, only the first few characters from each level are used. By default, `datadensity` will construct one axis (i.e., one strip) per variable in the data frame. Variable names appear to the left of the axes, and the number of missing values (if greater than zero) appear to the right of the axes. An optional group variable can be used for stratification, where the different strata are depicted using different colors. If the `q` vector is specified, the desired quantiles (over all groups) are displayed with solid triangles below each axis.

When the sample size exceeds 2000 (this value may be modified using the `nhistSpike` argument, `datadensity` calls `histSpike` instead of `scat1d` to show the data density for numeric variables. This results in a histogram-like display that makes the resulting graphics file much smaller. In this case, `datadensity` uses the `minf` argument (see below) so that very infrequent data values will not be lost on the variable's axis, although this will slightly distort the histogram.

`histSpike` is another method for showing a high-resolution data distribution that is particularly good for very large datasets (say  $n > 1000$ ). By default, `histSpike` bins the continuous `x` variable into 100 equal-width bins and then computes the frequency counts within bins (if `n` does not exceed 10, no binning is done). If `add=FALSE` (the default), the function displays either proportions or frequencies as in a vertical histogram. Instead of bars, spikes are used to depict the frequencies. If `add=TRUE`, the function assumes you are adding small density displays that are intended to take up a small amount of space in the margins of the overall plot. The `frac` argument is used as with `scat1d` to determine the relative length of the whole plot that is used to represent the maximum frequency. No jittering is done by `histSpike`.

`histSpike` can also graph a kernel density estimate for `x`, or add a small density curve to any of 4 sides of an existing plot. When `y` or `curve` is specified, the density or spikes are drawn with respect to the curve rather than the `x`-axis.

`histSpikeg` is similar to `histSpike` but is for adding layers to a `ggplot2` graphics object or traces to a `plotly` object. `histSpikeg` can also add `lowess` curves to the plot.

`ecdfpM` makes a `plotly` graph or series of graphs showing possibly superposed empirical cumulative distribution functions.

## Usage

```
scat1d(x, side=3, frac=0.02, jitfrac=0.008, tfrac,
      eps=ifelse(preserve,0,.001),
      lwd=0.1, col=par("col"),
      y=NULL, curve=NULL,
      bottom.align=FALSE,
      preserve=FALSE, fill=1/3, limit=TRUE, nhistSpike=2000, nint=100,
      type=c('proportion','count','density'), grid=FALSE, ...)
```

```
jitter2(x, ...)
```

```
## Default S3 method:
jitter2(x, fill=1/3, limit=TRUE, eps=0,
      presorted=FALSE, ...)
```



```

## S3 method for class 'data.frame'
jitter2(x, ...)

datadensity(object, ...)

## S3 method for class 'data.frame'
datadensity(object, group,
             which=c("all", "continuous", "categorical"),
             method.cat=c("bar", "freq"),
             col.group=1:10,
             n.unique=10, show.na=TRUE, nint=1, naxes,
             q, bottom.align=nint>1,
             cex.axis=sc(.5,.3), cex.var=sc(.8,.3),
             lmgp=NULL, tck=sc(-.009,-.002),
             ranges=NULL, labels=NULL, ...)
# sc(a,b) means default to a if number of axes <= 3, b if >=50, use
# linear interpolation within 3-50

histSpike(x, side=1, nint=100, bins=NULL, frac=.05, minf=NULL, mult.width=1,
          type=c('proportion', 'count', 'density'),
          xlim=range(x), ylim=c(0,max(f)), xlab=deparse(substitute(x)),
          ylab=switch(type,proportion='Proportion',
                     count      ='Frequency',
                     density    ='Density'),
          y=NULL, curve=NULL, add=FALSE, minimal=FALSE,
          bottom.align=type=='density', col=par('col'), lwd=par('lwd'),
          grid=FALSE, ...)

histSpikeg(formula=NULL, predictions=NULL, data, plotly=NULL,
           lowess=FALSE, xlim=NULL, ylim=NULL,
           side=1, nint=100,
           frac=function(f) 0.01 + 0.02*sqrt(f-1)/sqrt(max(f,2)-1),
           span=3/4, histcol='black', showlegend=TRUE)

ecdfpM(x, group=NULL, what=c('F', '1-F', 'f', '1-f'), q=NULL,
       extra=c(0.025, 0.025), xlab=NULL, ylab=NULL, height=NULL, width=NULL,
       colors=NULL, nrows=NULL, ncols=NULL, ...)

```

### Arguments

x	a vector of numeric data, or a data frame (for jitter2 or ecdfpM)
object	a data frame or list (even with unequal number of observations per variable, as long as group is not specified)
side	axis side to use (1=bottom (default for histSpike), 2=left, 3=top (default for scat1d), 4=right)
frac	fraction of smaller of vertical and horizontal axes for tick mark lengths. Can be negative to move tick marks outside of plot. For histSpike, this is the relative y-direction length to be used for the largest frequency. When scat1d calls

	histSpike, it multiplies its frac argument by 2.5. For histSpikeg, frac is a function of f, the vector of all frequencies. The default function scales tick marks so that they are between 0.01 and 0.03 of the y range, linearly scaled in the square root of the frequency less one.
jitfrac	fraction of axis for jittering. If $\text{jitfrac} \leq 0$ , no jittering is done. If <code>preserve=TRUE</code> , the amount of jittering is independent of <code>jitfrac</code> .
tfrac	Fraction of tick mark to actually draw. If $\text{tfrac} < 1$ , will draw a random fraction <code>tfrac</code> of the line segment at each point. This is useful for very large samples or ones with some very dense points. The default value is 1 if the number of non-missing observations <code>n</code> is less than 125, and $\max(.1, 125/n)$ otherwise.
eps	fraction of axis for determining overlapping points in <code>x</code> . For <code>preserve=TRUE</code> the default is 0 and original unique values are retained, bigger values of <code>eps</code> tends to bias observations from dense to sparse regions, but ranks are still preserved.
lwd	line width for tick marks, passed to <code>segments</code>
col	color for tick marks, passed to <code>segments</code>
y	specify a vector the same length as <code>x</code> to draw tick marks along a curve instead of by one of the axes. The <code>y</code> values are often predicted values from a model. The side argument is ignored when <code>y</code> is given. If the curve is already represented as a table look-up, you may specify it using the <code>curve</code> argument instead. <code>y</code> may be a scalar to use a constant vertical placement.
curve	a list containing elements <code>x</code> and <code>y</code> for which linear interpolation is used to derive <code>y</code> values corresponding to values of <code>x</code> . This results in tick marks being drawn along the curve. For <code>histSpike</code> , interpolated <code>y</code> values are derived for binmid-points.
minimal	for <code>histSpike</code> set <code>minimal=TRUE</code> to draw a minimalist spike histogram with no <code>y</code> -axis. This works best when produce graphics images that are short, e.g., have a height of two inches. <code>add</code> is forced to be <code>FALSE</code> in this case so that a standalone graph is produced. Only base graphics are used.
bottom.align	set to <code>TRUE</code> to have the bottoms of tick marks (for <code>side=1</code> or <code>side=3</code> ) aligned at the <code>y</code> -coordinate. The default behavior is to center the tick marks. For <code>datadensity.data.frame</code> , <code>bottom.align</code> defaults to <code>TRUE</code> if <code>nint&gt;1</code> . In other words, if you are only labeling the first and last axis tick mark, the <code>scat1d</code> tick marks are centered on the variable's axis.
preserve	set to <code>TRUE</code> to invoke <code>jitter2</code>
fill	maximum fraction of the axis filled by jittered values. If <code>d</code> are duplicated values between a lower value <code>l</code> and upper value <code>u</code> , then <code>d</code> will be spread within $\pm \text{fill} * \min(u - d, d - l)/2$ .
limit	specifies a limit for maximum shift in jittered values. Duplicate values will be spread within $\pm \text{fill} * \min(u - d, d - l)/2$ . The default <code>TRUE</code> restricts jittering to the smallest $\min(u - d, d - l)/2$ observed and results in equal amount of jittering for all <code>d</code> . Setting to <code>FALSE</code> allows for locally different amount of jittering, using maximum space available.
nhistSpike	If the number of observations exceeds or equals <code>nhistSpike</code> , <code>scat1d</code> will automatically call <code>histSpike</code> to draw the data density, to prevent the graphics file from being too large.

type	used by or passed to histSpike. Set to "count" to display frequency counts rather than relative frequencies, or "density" to display a kernel density estimate computed using the density function.
grid	set to TRUE if the R grid package is in effect for the current plot
nint	number of intervals to divide each continuous variable's axis for datadensity. For histSpike, is the number of equal-width intervals for which to bin x, and if instead nint is a character string (e.g., nint="all"), the frequency tabulation is done with no binning. In other words, frequencies for all unique values of x are derived and plotted. For histSpikeg, if x has no more than nint unique values, all observed values are used, otherwise the data are rounded before tabulation so that there are no more than nint intervals. For histSpike, nint is ignored if bins is given.
bins	for histSpike specifies the actual cutpoints to use for binning x. The default is to use nint in conjunction with xlim.
...	optional arguments passed to scat1d from datadensity or to histSpike from scat1d. For histSpikeg are passed to the lines list to add_trace. For ecdfpM these arguments are passed to add_lines.
presorted	set to TRUE to prevent from sorting for determining the order $l < d < u$ . This is useful if an existing meaningful local order would be destroyed by sorting, as in $\sin(\pi * \text{sort}(\text{round}(\text{runif}(1000, 0, 10), 1)))$ .
group	an optional stratification variable, which is converted to a factor vector if it is not one already
which	set which="continuous" to only plot continuous variables, or which="categorical" to only plot categorical, character, or discrete numeric ones. By default, all types of variables are depicted.
method.cat	set method.cat="freq" to depict frequencies of categorical variables with digits representing the cell frequencies, with size proportional to the square root of the frequency. By default, vertical bars are used.
col.group	colors representing the group strata. The vector of colors is recycled to be the same length as the levels of group.
n.unique	number of unique values a numeric variable must have before it is considered to be a continuous variable
show.na	set to FALSE to suppress drawing the number of NAs to the right of each axis
naxes	number of axes to draw on each page before starting a new plot. You can set naxes larger than the number of variables in the data frame if you want to compress the plot vertically.
q	a vector of quantiles to display. By default, quantiles are not shown.
extra	a two-vector specifying the fraction of the x range to add on the left and the fraction to add on the right
cex.axis	character size for draw labels for axis tick marks
cex.var	character size for variable names and frequency of NAs
lmgp	spacing between numeric axis labels and axis (see par for mgp)
tck	see tck under <a href="#">par</a>

ranges	a list containing ranges for some or all of the numeric variables. If ranges is not given or if a certain variable is not found in the list, the empirical range, modified by pretty, is used. Example: ranges=list(age=c(10,100), pressure=c(50,150)).
labels	a vector of labels to use in labeling the axes for datadensity.data.frame. Default is to use the names of the variable in the input data frame. Note: margin widths computed for setting aside names of variables use the names, and not these labels.
minf	For histSpike, if minf is specified low bin frequencies are set to a minimum value of minf times the maximum bin frequency, so that rare data points will remain visible. A good choice of minf is 0.075. datadensity.data.frame passes minf=0.075 to scat1d to pass to histSpike. Note that specifying minf will cause the shape of the histogram to be distorted somewhat.
mult.width	multiplier for the smoothing window width computed by histSpike when type="density"
xlim	a 2-vector specifying the outer limits of x for binning (and plotting, if add=FALSE and nint is a number). For histSpikeg, observations outside the xlim range are ignored.
ylim	y-axis range for plotting (if add=FALSE). Often needed for histSpikeg to help scale the tick mark line segments.
xlab	x-axis label (add=FALSE or for ecdfpM); default is name of input argument, or for ecdfpM comes from label and units attributes of the analysis variable. For ecdfpM xlab may be a vector if there is more than one analysis variable.
ylab	y-axis label (add=FALSE or for ecdfpM)
add	set to TRUE to add the spike-histogram to an existing plot, to show marginal data densities
formula	a formula of the form $y \sim x_1$ or $y \sim x_1 + \dots$ where y is the name of the y-axis variable being plotted with ggplot, x1 is the name of the x-axis variable, and optional ... are variables used by ggplot to produce multiple curves on a panel and/or facets.
predictions	the data frame being plotted by ggplot, containing x and y coordinates of curves. If omitted, spike histograms are drawn at the bottom (default) or top of the plot according to side.
data	for histSpikeg is a mandatory data frame containing raw data whose frequency distribution is to be summarized, using variables in formula.
plotly	an existing plotly object. If not NULL, histSpikeg uses plotly instead of ggplot.
lowess	set to TRUE to have histSpikeg add a geom_line layer to the ggplot2 graphic, containing lowess() nonparametric smoothers. This causes the returned value of histSpikeg to be a list with two components: "hist" and "lowess" each containing a layer. Fortunately, ggplot2 plots both layers automatically. If the dependent variable is binary, iter=0 is passed to lowess so that outlier detection is turned off; otherwise iter=3 is passed.
span	passed to lowess as the f argument
histcol	color of line segments (tick marks) for histSpikeg. Default is black. Set to any color or to "default" to use the prevailing colors for the graphic.

showlegend	set to FALSE too have the added plotly traces not have entries in the plot legend
what	set to "1-F" to plot 1 minus the ECDF instead of the ECDF, "f" to plot cumulative frequency, or "1-f" to plot the inverse cumulative frequency
height, width	passed to plot_ly
colors	a vector of colors to pas to add_lines
nrows, ncols	passed to plotly::subplot

### Details

For `scat1d` the length of line segments used is  $\text{frac} \times \min(\text{par()} \$ \text{pin}) / \text{par()} \$ \text{uin}[\text{opp}]$  data units, where `opp` is the index of the opposite axis and `frac` defaults to `.02`. Assumes that `plot` has already been called. Current `par("usr")` is used to determine the range of data for the axis of the current plot. This range is used in jittering and in constructing line segments.

### Value

`histSpike` returns the actual range of `x` used in its binning

### Side Effects

`scat1d` adds line segments to `plot.datadensity.data.frame` draws a complete plot. `histSpike` draws a complete plot or adds to an existing plot.

### Author(s)

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 Nashville TN, USA  
 <fh@fharrell.com>

Martin Maechler (improved `scat1d`)  
 Seminar fuer Statistik  
 ETH Zurich SWITZERLAND  
 <maechler@stat.math.ethz.ch>

Jens Oehlschlaegel-Akiyoshi (wrote `jitter2`)  
 Center for Psychotherapy Research  
 Christian-Belser-Strasse 79a  
 D-70597 Stuttgart Germany  
 <oehl@psyres-stuttgart.de>

### See Also

[segments](#), [jitter](#), [rug](#), [plsmo](#), [lowess](#), [stripplot](#), [hist.data.frame](#), [Ecdf](#), [hist](#), [histogram](#), [table](#), [density](#), [stat\\_plsmo](#), [histboxp](#)

**Examples**

```

plot(x <- rnorm(50), y <- 3*x + rnorm(50)/2 )
scat1d(x)          # density bars on top of graph
scat1d(y, 4)      # density bars at right
histSpike(x, add=TRUE)      # histogram instead, 100 bins
histSpike(y, 4, add=TRUE)
histSpike(x, type='density', add=TRUE) # smooth density at bottom
histSpike(y, 4, type='density', add=TRUE)

smooth <- lowess(x, y) # add nonparametric regression curve
lines(smooth)         # Note: plsmo() does this
scat1d(x, y=approx(smooth, xout=x)$y) # data density on curve
scat1d(x, curve=smooth) # same effect as previous command
histSpike(x, curve=smooth, add=TRUE) # same as previous but with histogram
histSpike(x, curve=smooth, type='density', add=TRUE)
# same but smooth density over curve

plot(x <- rnorm(250), y <- 3*x + rnorm(250)/2)
scat1d(x, tfrac=0)      # dots randomly spaced from axis
scat1d(y, 4, frac=-.03) # bars outside axis
scat1d(y, 2, tfrac=.2) # same bars with smaller random fraction

x <- c(0:3,rep(4,3),5,rep(7,10),9)
plot(x, jitter2(x))    # original versus jittered values
abline(0,1)           # unique valuesunjittered on abline
points(x+0.1, jitter2(x, limit=FALSE), col=2)
# allow locally maximum jittering
points(x+0.2, jitter2(x, fill=1), col=3); abline(h=seq(0.5,9,1), lty=2)
# fill 3/3 instead of 1/3

x <- rnorm(200,0,2)+1; y <- x^2
x2 <- round((x+rnorm(200))/2)*2
x3 <- round((x+rnorm(200))/4)*4
dfram <- data.frame(y,x,x2,x3)
plot(dfram$x2, dfram$y) # jitter2 via scat1d
scat1d(dfram$x2, y=dfram$y, preserve=TRUE, col=2)
scat1d(dfram$x2, preserve=TRUE, frac=-0.02, col=2)
scat1d(dfram$y, 4, preserve=TRUE, frac=-0.02, col=2)

pairs(jitter2(dfram)) # pairs for jittered data.frame
# This gets reasonable pairwise scatter plots for all combinations of
# variables where
#
# - continuous variables (with unique values) are not jittered at all, thus
#   all relations between continuous variables are shown as they are,
#   extreme values have exact positions.
#
# - discrete variables get a reasonable amount of jittering, whether they
#   have 2, 3, 5, 10, 20 \dots levels

```

```

#
# - different from adding noise, jitter2() will use the available space
#   optimally and no value will randomly mask another
#
# If you want a scatterplot with lowess smooths on the *exact* values and
# the point clouds shown jittered, you just need
#
pairs( dfram ,panel=function(x,y) { points(jitter2(x),jitter2(y))
                                     lines(lowess(x,y)) } )

datadensity(dfram)      # graphical snapshot of entire data frame
datadensity(dfram, group=cut2(dfram$x2,g=3))
                        # stratify points and frequencies by
                        # x2 tertiles and use 3 colors

# datadensity.data.frame(split(x, grouping.variable))
# need to explicitly invoke datadensity.data.frame when the
# first argument is a list

## Not run:
require(rms)
require(ggplot2)
f <- lrm(y ~ blood.pressure + sex * (age + rcs(cholesterol,4)),
        data=d)
p <- Predict(f, cholesterol, sex)
g <- ggplot(p, aes(x=cholesterol, y=yhat, color=sex)) + geom_line() +
  xlab(xl2) + ylim(-1, 1)
g <- g + geom_ribbon(data=p, aes(ymin=lower, ymax=upper), alpha=0.2,
                    linetype=0, show_guide=FALSE)
g + histSpikeg(yhat ~ cholesterol + sex, p, d)

# colors <- c('red', 'blue')
# p <- plot_ly(x=x, y=y, color=g, colors=colors, mode='markers')
# histSpikep(p, x, y, z, color=g, colors=colors)

w <- data.frame(x1=rnorm(100), x2=exp(rnorm(100)))
g <- c(rep('a', 50), rep('b', 50))
ecdfpM(w, group=g, ncols=2)

## End(Not run)

```

**Description**

Creates a new variable from a series of logical conditions. The new variable can be a hierarchical category or score derived from considering the rightmost TRUE value among the input variables, an additive point score, a union, or any of several others by specifying a function using the fun argument.

**Usage**

```
score.binary(..., fun=max, points=1:p,
             na.rm=funtext == "max", retfactor=TRUE)
```

**Arguments**

...	a list of variables or expressions which are considered to be binary or logical
fun	a function to compute on each row of the matrix represented by a specific observation of all the variables in ...
points	points to assign to successive elements of ... . The default is 1, 2, ..., p, where p is the number of elements. If you specify one number for points, that number will be duplicated (i.e., equal weights are assumed).
na.rm	set to TRUE to remove NAs from consideration when processing each row of the matrix of variables in ... . For fun=max, na.rm=TRUE is the default since score.binary assumes that a hierarchical scale is based on available information. Otherwise, na.rm=FALSE is assumed. For fun=mean you may want to specify na.rm=TRUE.
retfactor	applies if fun=max, in which case retfactor=TRUE makes score.binary return a factor object since a hierarchical scale implies a unique choice.

**Value**

a factor object if retfactor=TRUE and fun=max or a numeric vector otherwise. Will not contain NAs if na.rm=TRUE unless every variable in a row is NA. If a factor object is returned, it has levels "none" followed by character string versions of the arguments given in ... .

**See Also**

[any](#), [sum](#), [max](#), [factor](#)

**Examples**

```
set.seed(1)
age <- rnorm(25, 70, 15)
previous.disease <- sample(0:1, 25, TRUE)
#Hierarchical scale, highest of 1:age>70 2:previous.disease
score.binary(age>70, previous.disease, retfactor=FALSE)
#Same as above but return factor variable with levels "none" "age>70"
# "previous.disease"
score.binary(age>70, previous.disease)
```



```
#Additive scale with weights 1:age>70 2:previous.disease
score.binary(age>70, previous.disease, fun=sum)
#Additive scale, equal weights
score.binary(age>70, previous.disease, fun=sum, points=c(1,1))
#Same as saying points=1

#Union of variables, to create a new binary variable
score.binary(age>70, previous.disease, fun=any)
```

---

sedit	<i>Character String Editing and Miscellaneous Character Handling Functions</i>
-------	--

---

## Description

This suite of functions was written to implement many of the features of the UNIX sed program entirely within S (function `sedit`). The `substring.location` function returns the first and last position numbers that a sub-string occupies in a larger string. The `substring2<-` function does the opposite of the builtin function `substring`. It is named `substring2` because for S-Plus there is a built-in function `substring`, but it does not handle multiple replacements in a single string. `replace.substring.wild` edits character strings in the fashion of "change xxxxANYTHINGyyyy to aaaaANYTHINGbbbb", if the "ANYTHING" passes an optional user-specified test function. Here, the "yyyy" string is searched for from right to left to handle balancing parentheses, etc. `numeric.string` and `all.digits` are two examples of test functions, to check, respectively if each of a vector of strings is a legal numeric or if it contains only the digits 0-9. For the case where `old="*$"` or `"^*"`, or for `replace.substring.wild` with the same values of `old` or with `front=TRUE` or `back=TRUE`, `sedit` (if `wild.literal=FALSE`) and `replace.substring.wild` will edit the largest substring satisfying test.

`substring2` is just a copy of `substring` so that `substring2<-` will work.

## Usage

```
sedit(text, from, to, test, wild.literal=FALSE)
substring.location(text, string, restrict)
# substring(text, first, last) <- setto # S-Plus only
replace.substring.wild(text, old, new, test, front=FALSE, back=FALSE)
numeric.string(string)
all.digits(string)
substring2(text, first, last)
substring2(text, first, last) <- value
```

## Arguments

text	a vector of character strings for <code>sedit</code> , <code>substring2</code> , <code>substring2&lt;-</code> or a single character string for <code>substring.location</code> , <code>replace.substring.wild</code> .
------	--

from	a vector of character strings to translate from, for <code>sedit</code> . A single asterisk wild card, meaning allow any sequence of characters (subject to the test function, if any) in place of the <code>"*"</code> . An element of <code>from</code> may begin with <code>"^"</code> to force the match to begin at the beginning of text, and an element of <code>from</code> can end with <code>"\$"</code> to force the match to end at the end of text.
to	a vector of character strings to translate to, for <code>sedit</code> . If a corresponding element in <code>from</code> had an <code>"*"</code> , the element in <code>to</code> may also have an <code>"*"</code> . Only single asterisks are allowed. If <code>to</code> is not the same length as <code>from</code> , the <code>rep</code> function is used to make it the same length.
string	a single character string, for <code>substring.location</code> , <code>numeric.string</code> , <code>all.digits</code>
first	a vector of integers specifying the first position to replace for <code>substring2&lt;-</code> . <code>first</code> may also be a vector of character strings that are passed to <code>sedit</code> to use as patterns for replacing substrings with <code>setto</code> . See one of the last examples below.
last	a vector of integers specifying the ending positions of the character substrings to be replaced. The default is to go to the end of the string. When <code>first</code> is character, <code>last</code> must be omitted.
setto	a character string or vector of character strings used as replacements, in <code>substring2&lt;-</code>
old	a character string to translate from for <code>replace.substring.wild</code> . May be <code>"*\$"</code> or <code>"^*"</code> or any string containing a single <code>"*"</code> but not beginning with <code>"^"</code> or ending with <code>"\$"</code> .
new	a character string to translate to for <code>replace.substring.wild</code>
test	a function of a vector of character strings returning a logical vector whose elements are TRUE or FALSE according to whether that string element qualifies as the wild card string for <code>sedit</code> , <code>replace.substring.wild</code>
wild.literal	set to TRUE to not treat asterisks as wild cards and to not look for <code>"^"</code> or <code>"\$"</code> in <code>old</code>
restrict	a vector of two integers for <code>substring.location</code> which specifies a range to which the search for matches should be restricted
front	specifying <code>front = TRUE</code> and <code>old = "*" is the same as specifying <code>old = "^*"</code></code>
back	specifying <code>back = TRUE</code> and <code>old = "*" is the same as specifying <code>old = "*\$"</code></code>
value	a character vector

### Value

`sedit` returns a vector of character strings the same length as `text`. `substring.location` returns a list with components named `first` and `last`, each specifying a vector of character positions corresponding to matches. `replace.substring.wild` returns a single character string. `numeric.string` and `all.digits` return a single logical value.

### Side Effects

`substring2<-` modifies its first argument

**Author(s)**

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University School of Medicine  
 <fh@fharrell.com>

**See Also**

[grep](#), [substring](#)

**Examples**

```
x <- 'this string'
substring2(x, 3, 4) <- 'IS'
x
substring2(x, 7) <- ''
x
```

```
substring.location('abcdefgabc', 'ab')
substring.location('abcdefgabc', 'ab', restrict=c(3,999))
```

```
replace.substring.wild('this is a cat','this*cat','that*dog')
replace.substring.wild('there is a cat','is a*', 'is not a*')
replace.substring.wild('this is a cat','is a*', 'Z')
```

```
qualify <- function(x) x==' 1.5 ' | x==' 2.5 '
replace.substring.wild('He won 1.5 million $','won*million',
  'lost*million', test=qualify)
replace.substring.wild('He won 1 million $','won*million',
  'lost*million', test=qualify)
replace.substring.wild('He won 1.2 million $','won*million',
  'lost*million', test=numeric.string)
```

```
x <- c('a = b','c < d','hello')
sedit(x, c('=','he*o'),c('==','he*'))
```

```
sedit('x23', '*$', '[*]', test=numeric.string)
sedit('23xx', '^*', 'Y_{*}', test=all.digits)
```

```
replace.substring.wild("abcdefabcdef", "d*f", "xy")
```

```
x <- "abcd"
substring2(x, "bc") <- "BCX"
x
substring2(x, "B*d") <- "B*D"
```

x

seqFreq

*seqFreq***Description**

Find Sequential Exclusions Due to NAs

**Usage**

```
seqFreq(..., labels = NULL, noneNA = FALSE)
```

**Arguments**

...	any number of variables
labels	if specified variable labels will be used in place of variable names
noneNA	set to TRUE to not include 'none' as a level in the result

**Details**

Finds the variable with the highest number of NAs. From the non-NAs on that variable find the next variable from those remaining with the highest number of NAs. Proceed in like fashion. The resulting variable summarizes sequential exclusions in a hierarchical fashion. See [this](#) for more information.

**Value**

factor variable with obs.per.numcond attribute

**Author(s)**

Frank Harrell

show.pch

*Display Colors, Plotting Symbols, and Symbol Numeric Equivalent***Description**

show.pch plots the definitions of the pch parameters. show.col plots definitions of integer-valued colors. character.table draws numeric equivalents of all latin characters; the character on line xy and column z of the table has numeric code "xyz", which you would surround in quotes and precede by a backslash.

**Usage**

```
show.pch(object = par("font"))
show.col(object=NULL)
character.table(font=1)
```

**Arguments**

object	font for show.pch, ignored for show.col.
font	font

**Author(s)**

Pierre Joyet <pierre.joyet@bluewin.ch>, Frank Harrell

**See Also**

[points](#), [text](#)

**Examples**

```
## Not run:
show.pch()
show.col()
character.table()

## End(Not run)
```

---

showPsfrag

*Display image from psfrag LaTeX strings*

---

**Description**

showPsfrag is used to display (using ghostview) a postscript image that contained psfrag LaTeX strings, by building a small LaTeX script and running latex and dvips.

**Usage**

```
showPsfrag(filename)
```

**Arguments**

filename	name or character string or character vector specifying file prefix.
----------	--

**Author(s)**

Frank Harrell  
Department of Biostatistics  
Vanderbilt University  
<fh@fharrell.com>

**References**

Grant MC, Carlisle (1998): The PSfrag System, Version 3. Full documentation is obtained by searching [www.ctan.org](http://www.ctan.org) for 'pfgguide.ps'.

**See Also**

[postscript](#), [par](#), [ps.options](#), [mgp.axis.labels](#), [pdf](#), [trellis.device](#), [setTrellis](#)

---

 simMarkovOrd

*simMarkovOrd*


---

**Description**

Simulate Ordinal Markov Process

**Usage**

```
simMarkovOrd(
  n = 1,
  y,
  times,
  initial,
  X = NULL,
  absorb = NULL,
  intercepts,
  g,
  carry = FALSE,
  rdsample = NULL,
  ...
)
```

**Arguments**

n	number of subjects to simulate
y	vector of possible y values in order (numeric, character, factor)
times	vector of measurement times
initial	initial value of y (baseline state; numeric, character, or factor matching y). If length 1 this value is used for all subjects, otherwise it is a vector of length n.
X	an optional vector of matrix of baseline covariate values passed to g. If a vector, X represents a set of single values for all the covariates and those values are used for every subject. Otherwise X is a matrix with rows corresponding to subjects and columns corresponding to covariates which g must know how to handle. g only sees one row of X at a time.
absorb	vector of absorbing states, a subset of y (numeric, character, or factor matching y). The default is no absorbing states. Observations are truncated when an absorbing state is simulated.

intercepts	vector of intercepts in the proportional odds model. There must be one fewer of these than the length of <i>y</i> .
g	a user-specified function of three or more arguments which in order are <i>yprev</i> - the value of <i>y</i> at the previous time, the current time <i>t</i> , the gap between the previous time and the current time, an optional (usually named) covariate vector <i>X</i> , and optional arguments such as a regression coefficient value to simulate from. The function needs to allow <i>yprev</i> to be a vector and <i>yprev</i> must not include any absorbing states. The <i>g</i> function returns the linear predictor for the proportional odds model aside from <i>intercepts</i> . The returned value must be a matrix with row names taken from <i>yprev</i> . If the model is a proportional odds model, the returned value must be one column. If it is a partial proportional odds model, the value must have one column for each distinct value of the response variable <i>Y</i> after the first one, with the levels of <i>Y</i> used as optional column names. So columns correspond to <i>intercepts</i> . The different columns are used for <i>y</i> -specific contributions to the linear predictor (aside from <i>intercepts</i> ) for a partial or constrained partial proportional odds model. Parameters for partial proportional odds effects may be included in the ... arguments.
carry	set to TRUE to carry absorbing state forward after it is first hit; the default is to end records for the subject once the absorbing state is hit
rdsample	an optional function to do response-dependent sampling. It is a function of these arguments, which are vectors that stop at any absorbing state: <i>times</i> (ascending measurement times for one subject), <i>y</i> (vector of ordinal outcomes at these times for one subject). The function returns NULL if no observations are to be dropped, returns the vector of new times to sample.
...	additional arguments to pass to <i>g</i> such as a regression coefficient

### Details

Simulates longitudinal data for subjects following a first-order Markov process under a proportional odds model. Optionally, response-dependent sampling can be done, e.g., if a subject hits a specified state at time *t*, measurements are removed for times *t*+1, *t*+3, *t*+5, ... This is applicable when for example a study of hospitalized patients samples every day, *Y*=1 denotes patient discharge to home, and sampling is less frequent outside the hospital. This example assumes that arriving home is not an absorbing state, i.e., a patient could return to the hospital.

### Value

data frame with one row per subject per time, and columns *id*, *time*, *yprev*, *y*, values in ...

### Author(s)

Frank Harrell

### See Also

<https://hbiostat.org/R/Hmisc/markov/>

---

`simplifyDims`*List Simplification*

---

**Description**

Takes a list where each element is a group of rows that have been spanned by a multirow row and combines it into one large matrix.

**Usage**

```
simplifyDims(x)
```

**Arguments**

`x` list of spanned rows

**Details**

All rows must have the same number of columns. This is used to format the list for printing.

**Value**

a matrix that contains all of the spanned rows.

**Author(s)**

Charles Dupont

**See Also**

[rbind](#)

**Examples**

```
a <- list(a = matrix(1:25, ncol=5), b = matrix(1:10, ncol=5), c = 1:5)
simplifyDims(a)
```



simRegOrd

*Simulate Power for Adjusted Ordinal Regression Two-Sample Test***Description**

This function simulates the power of a two-sample test from a proportional odds ordinal logistic model for a continuous response variable- a generalization of the Wilcoxon test. The continuous data model is normal with equal variance. Nonlinear covariate adjustment is allowed, and the user can optionally specify discrete ordinal level overrides to the continuous response. For example, if the main response is systolic blood pressure, one can add two ordinal categories higher than the highest observed blood pressure to capture heart attack or death.

**Usage**

```
simRegOrd(n, nsim=1000, delta=0, odds.ratio=1, sigma,
          p=NULL, x=NULL, X=x, Eyx, alpha=0.05, pr=FALSE)
```

**Arguments**

n	combined sample size (both groups combined)
nsim	number of simulations to run
delta	difference in means to detect, for continuous portion of response variable
odds.ratio	odds ratio to detect for ordinal overrides of continuous portion
sigma	standard deviation for continuous portion of response
p	a vector of marginal cell probabilities which must add up to one. The <i>i</i> th element specifies the probability that a patient will be in response level <i>i</i> for the control arm for the discrete ordinal overrides.
x	optional covariate to adjust for - a vector of length n
X	a design matrix for the adjustment covariate x if present. This could represent for example x and x <sup>2</sup> or cubic spline components.
Eyx	a function of x that provides the mean response for the control arm treatment
alpha	type I error
pr	set to TRUE to see iteration progress

**Value**

a list containing n, delta, sigma, power, betas, se, pvals where power is the estimated power (scalar), and betas, se, pvals are nsim-vectors containing, respectively, the ordinal model treatment effect estimate, standard errors, and 2-tailed p-values. When a model fit failed, the corresponding entries in betas, se, pvals are NA and power is the proportion of non-failed iterations for which the treatment p-value is significant at the alpha level.

**Author(s)**

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University School of Medicine  
 <fh@fharrell.com>

**See Also**

[popower](#)

**Examples**

```
## Not run:
## First use no ordinal high-end category overrides, and compare power
## to t-test when there is no covariate

n <- 100
delta <- .5
sd <- 1
require(pwr)
power.t.test(n = n / 2, delta=delta, sd=sd, type='two.sample') # 0.70
set.seed(1)
w <- simRegOrd(n, delta=delta, sigma=sd, pr=TRUE) # 0.686

## Now do ANCOVA with a quadratic effect of a covariate
n <- 100
x <- rnorm(n)
w <- simRegOrd(n, nsim=400, delta=delta, sigma=sd, x=x,
              X=cbind(x, x^2),
              Eyx=function(x) x + x^2, pr=TRUE)
w$power # 0.68

## Fit a cubic spline to some simulated pilot data and use the fitted
## function as the true equation in the power simulation
require(rms)
N <- 1000
set.seed(2)
x <- rnorm(N)
y <- x + x^2 + rnorm(N, 0, sd=sd)
f <- ols(y ~ rcs(x, 4), x=TRUE)

n <- 100
j <- sample(1 : N, n, replace=n > N)
x <- x[j]
X <- f$x[j,]
w <- simRegOrd(n, nsim=400, delta=delta, sigma=sd, x=x,
              X=X,
              Eyx=Function(f), pr=TRUE)
w$power ## 0.70

## Finally, add discrete ordinal category overrides and high end of y
## Start with no effect of treatment on these ordinal event levels (OR=1.0)
```

```

w <- simRegOrd(n, nsim=400, delta=delta, odds.ratio=1, sigma=sd,
              x=x, X=X, Eyx=Function(f),
              p=c(.98, .01, .01),
              pr=TRUE)
w$power ## 0.61 (0.3 if p=.8 .1 .1, 0.37 for .9 .05 .05, 0.50 for .95 .025 .025)

## Now assume that odds ratio for treatment is 2.5
## First compute power for clinical endpoint portion of Y alone
or <- 2.5
p <- c(.9, .05, .05)
popower(p, odds.ratio=or, n=100) # 0.275
## Compute power of t-test on continuous part of Y alone
power.t.test(n = 100 / 2, delta=delta, sd=sd, type='two.sample') # 0.70
## Note this is the same as the p.o. model power from simulation above
## Solve for OR that gives the same power estimate from popower
popower(rep(.01, 100), odds.ratio=2.4, n=100) # 0.706
## Compute power for continuous Y with ordinal override
w <- simRegOrd(n, nsim=400, delta=delta, odds.ratio=or, sigma=sd,
              x=x, X=X, Eyx=Function(f),
              p=c(.9, .05, .05),
              pr=TRUE)
w$power ## 0.72

## End(Not run)

```

---

smean.sd

---

*Compute Summary Statistics on a Vector*


---

## Description

A number of statistical summary functions is provided for use with `summary.formula` and `summarize` (as well as `tapply` and by themselves). `smean.cl.normal` computes 3 summary variables: the sample mean and lower and upper Gaussian confidence limits based on the t-distribution. `smean.sd` computes the mean and standard deviation. `smean.sdl` computes the mean plus or minus a constant times the standard deviation. `smean.cl.boot` is a very fast implementation of the basic nonparametric bootstrap for obtaining confidence limits for the population mean without assuming normality. These functions all delete NAs automatically. `smedian.hilow` computes the sample median and a selected pair of outer quantiles having equal tail areas.

## Usage

```
smean.cl.normal(x, mult=qt((1+conf.int)/2,n-1), conf.int=.95, na.rm=TRUE)
```

```
smean.sd(x, na.rm=TRUE)
```

```
smean.sdl(x, mult=2, na.rm=TRUE)
```

```
smean.cl.boot(x, conf.int=.95, B=1000, na.rm=TRUE, reps=FALSE)
```

```
smedian.hilow(x, conf.int=.95, na.rm=TRUE)
```

### Arguments

<code>x</code>	for summary functions <code>smean.*</code> , <code>smedian.hilow</code> , a numeric vector from which NAs will be removed automatically
<code>na.rm</code>	defaults to TRUE unlike built-in functions, so that by default NAs are automatically removed
<code>mult</code>	for <code>smean.cl.normal</code> is the multiplier of the standard error of the mean to use in obtaining confidence limits of the population mean (default is appropriate quantile of the t distribution). For <code>smean.sdl</code> , <code>mult</code> is the multiplier of the standard deviation used in obtaining a coverage interval about the sample mean. The default is <code>mult=2</code> to use plus or minus 2 standard deviations.
<code>conf.int</code>	for <code>smean.cl.normal</code> and <code>smean.cl.boot</code> specifies the confidence level (0-1) for interval estimation of the population mean. For <code>smedian.hilow</code> , <code>conf.int</code> is the coverage probability the outer quantiles should target. When the default, 0.95, is used, the lower and upper quantiles computed are 0.025 and 0.975.
<code>B</code>	number of bootstrap resamples for <code>smean.cl.boot</code>
<code>reps</code>	set to TRUE to have <code>smean.cl.boot</code> return the vector of bootstrapped means as the <code>reps</code> attribute of the returned object

### Value

a vector of summary statistics

### Author(s)

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>

### See Also

[summarize](#), [summary.formula](#)

### Examples

```
set.seed(1)
x <- rnorm(100)
smean.sd(x)
smean.sdl(x)
smean.cl.normal(x)
smean.cl.boot(x)
smedian.hilow(x, conf.int=.5) # 25th and 75th percentiles

# Function to compute 0.95 confidence interval for the difference in two means
# g is grouping variable
```

```

bootdif <- function(y, g) {
  g <- as.factor(g)
  a <- attr(smean.cl.boot(y[g==levels(g)[1]], B=2000, reps=TRUE), 'reps')
  b <- attr(smean.cl.boot(y[g==levels(g)[2]], B=2000, reps=TRUE), 'reps')
  meandif <- diff(tapply(y, g, mean, na.rm=TRUE))
  a.b <- quantile(b-a, c(.025,.975))
  res <- c(meandif, a.b)
  names(res) <- c('Mean Difference', '.025', '.975')
  res
}

```

---

 solvet

*solve Function with tol argument*


---

### Description

A slightly modified version of `solve` that allows a tolerance argument for singularity (`tol`) which is passed to `qr`.

### Usage

```
solvet(a, b, tol=1e-09)
```

### Arguments

<code>a</code>	a square numeric matrix
<code>b</code>	a numeric vector or matrix
<code>tol</code>	tolerance for detecting linear dependencies in columns of <code>a</code>

### See Also

[solve](#)

---

 somers2

*Somers' Dxy Rank Correlation*


---

### Description

Computes Somers' Dxy rank correlation between a variable `x` and a binary (0-1) variable `y`, and the corresponding receiver operating characteristic curve area `c`. Note that  $D_{xy} = 2(c - 0.5)$ . `somers` allows for a `weights` variable, which specifies frequencies to associate with each observation.

### Usage

```
somers2(x, y, weights=NULL, normwt=FALSE, na.rm=TRUE)
```

**Arguments**

<code>x</code>	typically a predictor variable. NAs are allowed.
<code>y</code>	a numeric outcome variable coded 0–1. NAs are allowed.
<code>weights</code>	a numeric vector of observation weights (usually frequencies). Omit or specify a zero-length vector to do an unweighted analysis.
<code>normwt</code>	set to TRUE to make <code>weights</code> sum to the actual number of non-missing observations.
<code>na.rm</code>	set to FALSE to suppress checking for NAs.

**Details**

The `rcorr.cens` function, which although slower than `somers2` for large sample sizes, can also be used to obtain `Dxy` for non-censored binary `y`, and it has the advantage of computing the standard deviation of the correlation index.

**Value**

a vector with the named elements `C`, `Dxy`, `n` (number of non-missing pairs), and `Missing`. Uses the formula  $C = (\text{mean}(\text{rank}(x)[y == 1]) - (n1 + 1)/2)/(n - n1)$ , where `n1` is the frequency of `y=1`.

**Author(s)**

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University School of Medicine  
 <fh@fharrell.com>

**See Also**

[concordance](#), [rcorr.cens](#), [rank](#), [wtd.rank](#),

**Examples**

```
set.seed(1)
predicted <- runif(200)
dead <- sample(0:1, 200, TRUE)
roc.area <- somers2(predicted, dead)["C"]

# Check weights
x <- 1:6
y <- c(0,0,1,0,1,1)
f <- c(3,2,2,3,2,1)
somers2(x, y)
somers2(rep(x, f), rep(y, f))
somers2(x, y, f)
```

---

soprobMarkovOrd	<i>soprobMarkovOrd</i>
-----------------	------------------------

---

**Description**

State Occupancy Probabilities for First-Order Markov Ordinal Model

**Usage**

```
soprobMarkovOrd(y, times, initial, absorb = NULL, intercepts, g, ...)
```

**Arguments**

<code>y</code>	a vector of possible <code>y</code> values in order (numeric, character, factor)
<code>times</code>	vector of measurement times
<code>initial</code>	initial value of <code>y</code> (baseline state; numeric, character, factor)
<code>absorb</code>	vector of absorbing states, a subset of <code>y</code> . The default is no absorbing states. (numeric, character, factor)
<code>intercepts</code>	vector of intercepts in the proportional odds model, with length one less than the length of <code>y</code>
<code>g</code>	a user-specified function of three or more arguments which in order are <code>yprev</code> - the value of <code>y</code> at the previous time, the current time <code>t</code> , the gap between the previous time and the current time, an optional (usually named) covariate vector <code>X</code> , and optional arguments such as a regression coefficient value to simulate from. The function needs to allow <code>yprev</code> to be a vector and <code>yprev</code> must not include any absorbing states. The <code>g</code> function returns the linear predictor for the proportional odds model aside from <code>intercepts</code> . The returned value must be a matrix with row names taken from <code>yprev</code> . If the model is a proportional odds model, the returned value must be one column. If it is a partial proportional odds model, the value must have one column for each distinct value of the response variable <code>Y</code> after the first one, with the levels of <code>Y</code> used as optional column names. So columns correspond to <code>intercepts</code> . The different columns are used for <code>y</code> -specific contributions to the linear predictor (aside from <code>intercepts</code> ) for a partial or constrained partial proportional odds model. Parameters for partial proportional odds effects may be included in the ... arguments.
<code>...</code>	additional arguments to pass to <code>g</code> such as covariate settings

**Value**

matrix with rows corresponding to `times` and columns corresponding to states, with values equal to exact state occupancy probabilities

**Author(s)**

Frank Harrell

**See Also**

<https://hbiostat.org/R/Hmisc/markov/>

---

soprobMarkovOrdm      *soprobMarkovOrdm*

---

**Description**

State Occupancy Probabilities for First-Order Markov Ordinal Model from a Model Fit

**Usage**

```
soprobMarkovOrdm(
  object,
  data,
  times,
  ylevels,
  absorb = NULL,
  tvarname = "time",
  pvarname = "yprev",
  gap = NULL
)
```

**Arguments**

object	a fit object created by <code>blrm</code> , <code>lrm</code> , <code>orm</code> , <code>VGAM::vglm()</code> , or <code>VGAM::vgam()</code>
data	a single observation list or data frame with covariate settings, including the initial state for Y
times	vector of measurement times
ylevels	a vector of ordered levels of the outcome variable (numeric or character)
absorb	vector of absorbing states, a subset of <code>ylevels</code> . The default is no absorbing states. (numeric, character, factor)
tvarname	name of time variable, defaulting to <code>time</code>
pvarname	name of previous state variable, defaulting to <code>yprev</code>
gap	name of time gap variable, defaults assuming that gap time is not in the model

**Details**

Computes state occupancy probabilities for a single setting of baseline covariates. If the model fit was from `rms::blrm()`, these probabilities are from all the posterior draws of the basic model parameters. Otherwise they are maximum likelihood point estimates.



**Value**

if object was not a Bayesian model, a matrix with rows corresponding to times and columns corresponding to states, with values equal to exact state occupancy probabilities. If object was created by `blrm`, the result is a 3-dimensional array with the posterior draws as the first dimension.

**Author(s)**

Frank Harrell

**See Also**

<https://hbiostat.org/R/Hmisc/markov/>

---

spikecomp

*spikecomp*

---

**Description**

Compute Elements of a Spike Histogram

**Usage**

```
spikecomp(
  x,
  method = c("tryactual", "simple", "grid"),
  lumptails = 0.01,
  normalize = TRUE,
  y,
  trans = NULL,
  result = c("list", "segments", "roundeddata")
)
```

**Arguments**

<code>x</code>	a numeric variable
<code>method</code>	specifies the binning and output method. The default is 'tryactual' and is intended to be used for spike histograms plotted in a way that allows for random x-coordinates and data gaps. No binning is done if there are less than 100 distinct values and the closest distinct x values are distinguishable (not with 1/500th of the data range of each other). Binning uses <code>pretty</code> . When <code>trans</code> is specified to transform <code>x</code> to reduce long tails due to outliers, <code>pretty</code> rounding is not done, and <code>lumptails</code> is ignored. <code>method='grid'</code> is intended for sparkline spike histograms drawn with bar charts, where plotting is done in a way that x-coordinates must be equally spaced. For this method, extensive binning information is returned. For either 'tryactual' or 'grid', the default if <code>trans</code> is omitted is to put all values beyond the 0.01 or 0.99 quantiles into a single bin so that outliers will not create long nearly empty tails. When <code>y</code> is specified, <code>method</code> is ignored.

<code>lumptails</code>	the quantile to use for lumping values into a single left and a single right bin for two of the methods. When outer quantiles using <code>lumptails</code> equal outer quantiles using <code>2*lumptails</code> , <code>lumptails</code> is ignored as this indicates a large number of ties in the tails of the distribution.
<code>normalize</code>	set to <code>FALSE</code> to not divide frequencies by maximum frequency
<code>y</code>	a vector of frequencies corresponding to <code>x</code> if you want the <code>(x, y)</code> pairs to be taken as a possibly irregular-spaced frequency tabulation for which you want to convert to a regularly-spaced tabulation like <code>count='tabulate'</code> produces. If there is a constant gap between <code>x</code> values, the original pairs are return, with possible removal of NAs.
<code>trans</code>	a list with three elements: the name of a transformation to make on <code>x</code> , the transformation function, and the inverse transformation function. The latter is used for <code>method='grid'</code> . When <code>trans</code> is given <code>lumptails</code> is ignored. <code>trans</code> applies only to <code>method='tryactual'</code> .
<code>tresult</code>	applies only to <code>method='tryactual'</code> . The default <code>'list'</code> returns a list with elements <code>x</code> , <code>y</code> , and <code>roundedTo</code> . <code>method='segments'</code> returns a list suitable for drawing line segments, with elements <code>x</code> , <code>y1</code> , <code>y2</code> . <code>method='roundeddata'</code> returns a list with elements <code>x</code> (non-tabulated rounded data vector after excluding NAs) and vector <code>roundedTo</code> .

## Details

Derives the line segment coordinates need to draw a spike histogram. This is useful for adding elements to `ggplot2` plots and for the `describe` function to construct spike histograms. Date/time variables are handled by doing calculations on the underlying numeric scale then converting back to the original class. For them the left endpoint of the first bin is taken as the minimal data value instead of rounded using `pretty()`.

## Value

when `y` is specified, a list with elements `x` and `y`. When `method='tryactual'` the returned value depends on `tresult`. For `method='grid'`, a list with elements `x` and `y` and scalar element `roundedTo` containing the typical bin width. Here `x` is a character string.

## Author(s)

Frank Harrell

## Examples

```
spikecomp(1:1000)
spikecomp(1:1000, method='grid')
## Not run:
On a data.table d use ggplot2 to make spike histograms by country and sex groups
s <- d[, spikecomp(x, tresult='segments'), by=(country, sex)]
ggplot(s) + geom_segment(aes(x=x, y=y1, xend=x, yend=y2, alpha=I(0.3))) +
  scale_y_continuous(breaks=NULL, labels=NULL) + ylab('') +
  facet_grid(country ~ sex)
```

```
## End(Not run)
```

---

```
spower          Simulate Power of 2-Sample Test for Survival under Complex Condi-
                tions
```

---

## Description

Given functions to generate random variables for survival times and censoring times, `spower` simulates the power of a user-given 2-sample test for censored data. By default, the logrank (Cox 2-sample) test is used, and a logrank function for comparing 2 groups is provided. Optionally a Cox model is fitted for each simulated dataset and the log hazard ratios are saved (this requires the `survival` package). A print method prints various measures from these. For composing R functions to generate random survival times under complex conditions, the `Quantile2` function allows the user to specify the intervention:control hazard ratio as a function of time, the probability of a control subject actually receiving the intervention (`dropin`) as a function of time, and the probability that an intervention subject receives only the control agent as a function of time (non-compliance, `dropout`). `Quantile2` returns a function that generates either control or intervention uncensored survival times subject to non-constant treatment effect, `dropin`, and `dropout`. There is a plot method for plotting the results of `Quantile2`, which will aid in understanding the effects of the two types of non-compliance and non-constant treatment effects. `Quantile2` assumes that the hazard function for either treatment group is a mixture of the control and intervention hazard functions, with mixing proportions defined by the `dropin` and `dropout` probabilities. It computes hazards and survival distributions by numerical differentiation and integration using a grid of (by default) 7500 equally-spaced time points.

The logrank function is intended to be used with `spower` but it can be used by itself. It returns the 1 degree of freedom chi-square statistic, with the associated Pike hazard ratio estimate as an attribute.

The `Weibull2` function accepts as input two vectors, one containing two times and one containing two survival probabilities, and it solves for the scale and shape parameters of the Weibull distribution ( $S(t) = e^{-\alpha t^\gamma}$ ) which will yield those estimates. It creates an R function to evaluate survival probabilities from this Weibull distribution. `Weibull2` is useful in creating functions to pass as the first argument to `Quantile2`.

The `Lognorm2` and `Gompertz2` functions are similar to `Weibull2` except that they produce survival functions for the log-normal and Gompertz distributions.

When `cox=TRUE` is specified to `spower`, the analyst may wish to extract the two margins of error by using the print method for `spower` objects (see example below) and take the maximum of the two.

## Usage

```
spower(rcontrol, rinterv, rcens, nc, ni,
       test=logrank, cox=FALSE, nsim=500, alpha=0.05, pr=TRUE)

## S3 method for class 'spower'
print(x, conf.int=.95, ...)

Quantile2(scontrol, hratio,
```

```

        dropin=function(times)0, dropout=function(times)0,
        m=7500, tmax, qtmx=.001, mplot=200, pr=TRUE, ...)

## S3 method for class 'Quantile2'
print(x, ...)

## S3 method for class 'Quantile2'
plot(x,
      what=c("survival", "hazard", "both", "drop", "hratio", "all"),
      dropsep=FALSE, lty=1:4, col=1, xlim, ylim=NULL,
      label.curves=NULL, ...)

logrank(S, group)

Gompertz2(times, surv)
Lognorm2(times, surv)
Weibull2(times, surv)

```

### Arguments

rcontrol	a function of n which returns n random uncensored failure times for the control group. spower assumes that non-compliance (dropin) has been taken into account by this function.
rinterv	similar to rcontrol but for the intervention group
rcens	a function of n which returns n random censoring times. It is assumed that both treatment groups have the same censoring distribution.
nc	number of subjects in the control group
ni	number in the intervention group
scontrol	a function of a time vector which returns the survival probabilities for the control group at those times assuming that all patients are compliant.
hratio	a function of time which specifies the intervention:control hazard ratio (treatment effect)
x	an object of class "Quantile2" created by Quantile2, or of class "spower" created by spower
conf.int	confidence level for determining fold-change margins of error in estimating the hazard ratio
S	a Surv object or other two-column matrix for right-censored survival times
group	group indicators have length equal to the number of rows in S argument.
times	a vector of two times
surv	a vector of two survival probabilities
test	any function of a Surv object and a grouping variable which computes a chi-square for a two-sample censored data test. The default is logrank.
cox	If true TRUE the two margins of error are available by using the print method for spower objects (see example below) and taking the maximum of the two.

<code>nsim</code>	number of simulations to perform (default=500)
<code>alpha</code>	type I error (default=.05)
<code>pr</code>	If FALSE prevents <code>power</code> from printing progress notes for simulations. If FALSE prevents <code>Quantile2</code> from printing <code>tmax</code> when it calculates <code>tmax</code> .
<code>dropin</code>	a function of time specifying the probability that a control subject actually is treated with the new intervention at the corresponding time
<code>dropout</code>	a function of time specifying the probability of an intervention subject dropping out to control conditions. As a function of time, <code>dropout</code> specifies the probability that a patient is treated with the control therapy at time <code>t</code> . <code>dropin</code> and <code>dropout</code> form mixing proportions for control and intervention hazard functions.
<code>m</code>	number of time points used for approximating functions (default is 7500)
<code>tmax</code>	maximum time point to use in the grid of <code>m</code> times. Default is the time such that <code>scontrol(time)</code> is <code>qtmax</code> .
<code>qtmax</code>	survival probability corresponding to the last time point used for approximating survival and hazard functions. Default is 0.001. For <code>qtmax</code> of the time for which a simulated time is needed which corresponds to a survival probability of less than <code>qtmax</code> , the simulated value will be <code>tmax</code> .
<code>mplot</code>	number of points used for approximating functions for use in plotting (default is 200 equally spaced points)
<code>...</code>	optional arguments passed to the <code>scontrol</code> function when it's evaluated by <code>Quantile2</code> . Unused for <code>print.power</code> .
<code>what</code>	a single character constant (may be abbreviated) specifying which functions to plot. The default is <code>"both"</code> meaning both survival and hazard functions. Specify <code>what="drop"</code> to just plot the <code>dropin</code> and <code>dropout</code> functions, <code>what="hratio"</code> to plot the hazard ratio functions, or <code>"all"</code> to make 4 separate plots showing all functions (6 plots if <code>dropsep=TRUE</code> ).
<code>dropsep</code>	If TRUE makes <code>plot.Quantile2</code> separate pure and contaminated functions onto separate plots
<code>lty</code>	vector of line types
<code>col</code>	vector of colors
<code>xlim</code>	optional x-axis limits
<code>ylim</code>	optional y-axis limits
<code>label.curves</code>	optional list which is passed as the <code>opts</code> argument to <a href="#">labcurve</a> .

### Value

`power` returns the power estimate (fraction of simulated chi-squares greater than the alpha-critical value). If `cox=TRUE`, `power` returns an object of class `"power"` containing the power and various other quantities.

`Quantile2` returns an R function of class `"Quantile2"` with attributes that drive the `plot` method. The major attribute is a list containing several lists. Each of these sub-lists contains a `Time` vector along with one of the following: survival probabilities for either treatment group and with or without contamination caused by non-compliance, hazard rates in a similar way, intervention:control hazard ratio function with and without contamination, and `dropin` and `dropout` functions.

logrank returns a single chi-square statistic and an attribute hr which is the Pike hazard ratio estimate.

Weibull2, Lognorm2 and Gompertz2 return an R function with three arguments, only the first of which (the vector of times) is intended to be specified by the user.

### Side Effects

spower prints the iteration number every 10 iterations if pr=TRUE.

### Author(s)

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University School of Medicine  
 <fh@fharrell.com>

### References

Lakatos E (1988): Sample sizes based on the log-rank statistic in complex clinical trials. *Biometrics* 44:229–241 (Correction 44:923).

Cuzick J, Edwards R, Segnan N (1997): Adjusting for non-compliance and contamination in randomized clinical trials. *Stat in Med* 16:1017–1029.

Cook, T (2003): Methods for mid-course corrections in clinical trials with survival outcomes. *Stat in Med* 22:3431–3447.

Barthel FMS, Babiker A et al (2006): Evaluation of sample size and power for multi-arm survival trials allowing for non-uniform accrual, non-proportional hazards, loss to follow-up and cross-over. *Stat in Med* 25:2521–2542.

### See Also

[cpower](#), [ciapower](#), [bpower](#), [cph](#), [coxph](#), [labcurve](#)

### Examples

```
# Simulate a simple 2-arm clinical trial with exponential survival so
# we can compare power simulations of logrank-Cox test with cpower()
# Hazard ratio is constant and patients enter the study uniformly
# with follow-up ranging from 1 to 3 years
# Drop-in probability is constant at .1 and drop-out probability is
# constant at .175. Two-year survival of control patients in absence
# of drop-in is .8 (mortality=.2). Note that hazard rate is -log(.8)/2
# Total sample size (both groups combined) is 1000
# % mortality reduction by intervention (if no dropin or dropout) is 25
# This corresponds to a hazard ratio of 0.7283 (computed by cpower)

cpower(2, 1000, .2, 25, accrual=2, tmin=1,
       noncomp.c=10, noncomp.i=17.5)
```

```

ranfun <- Quantile2(function(x)exp(log(.8)/2*x),
                   hratio=function(x)0.7283156,
                   dropin=function(x).1,
                   dropout=function(x).175)

rcontrol <- function(n) ranfun(n, what='control')
rinterv <- function(n) ranfun(n, what='int')
rcens <- function(n) runif(n, 1, 3)

set.seed(11) # So can reproduce results
spower(rcontrol, rinterv, rcens, nc=500, ni=500,
       test=logrank, nsim=50) # normally use nsim=500 or 1000

## Not run:
# Run the same simulation but fit the Cox model for each one to
# get log hazard ratios for the purpose of assessing the tightness
# confidence intervals that are likely to result

set.seed(11)
u <- spower(rcontrol, rinterv, rcens, nc=500, ni=500,
           test=logrank, nsim=50, cox=TRUE)
u
v <- print(u)
v[c('MOElower', 'MOEupper', 'SE')]

## End(Not run)

# Simulate a 2-arm 5-year follow-up study for which the control group's
# survival distribution is Weibull with 1-year survival of .95 and
# 3-year survival of .7. All subjects are followed at least one year,
# and patients enter the study with linearly increasing probability after that
# Assume there is no chance of dropin for the first 6 months, then the
# probability increases linearly up to .15 at 5 years
# Assume there is a linearly increasing chance of dropout up to .3 at 5 years
# Assume that the treatment has no effect for the first 9 months, then
# it has a constant effect (hazard ratio of .75)

# First find the right Weibull distribution for compliant control patients
sc <- Weibull2(c(1,3), c(.95,.7))
sc

# Inverse cumulative distribution for case where all subjects are followed
# at least a years and then between a and b years the density rises
# as (time - a) ^ d is a + (b-a) * u ^ (1/(d+1))

rcens <- function(n) 1 + (5-1) * (runif(n) ^ .5)
# To check this, type hist(rcens(10000), nclass=50)

```

```

# Put it all together

f <- Quantile2(sc,
  hratio=function(x)ifelse(x<=.75, 1, .75),
  dropin=function(x)ifelse(x<=.5, 0, .15*(x-.5)/(5-.5)),
  dropout=function(x).3*x/5)

par(mfrow=c(2,2))
# par(mfrow=c(1,1)) to make legends fit
plot(f, 'all', label.curves=list(keys='lines'))

rcontrol <- function(n) f(n, 'control')
rinterv <- function(n) f(n, 'intervention')

set.seed(211)
spower(rcontrol, rinterv, rcens, nc=350, ni=350,
  test=logrank, nsim=50) # normally nsim=500 or more
par(mfrow=c(1,1))

# Compose a censoring time generator function such that at 1 year
# 5% of subjects are accrued, at 3 years 70% are accrued, and at 10
# years 100% are accrued. The trial proceeds two years past the last
# accrual for a total of 12 years of follow-up for the first subject.
# Use linear interpolation between these 3 points

rcens <- function(n)
{
  times <- c(0,1,3,10)
  accrued <- c(0,.05,.7,1)
  # Compute inverse of accrued function at U(0,1) random variables
  accrual.times <- approx(accrued, times, xout=runif(n))$y
  censor.times <- 12 - accrual.times
  censor.times
}

censor.times <- rcens(500)
# hist(censor.times, nclass=20)
accrual.times <- 12 - censor.times
# Ecdf(accrual.times)
# lines(c(0,1,3,10), c(0,.05,.7,1), col='red')
# spower(..., rcens=rcens, ...)

## Not run:
# To define a control survival curve from a fitted survival curve
# with coordinates (tt, surv) with tt[1]=0, surv[1]=1:

Scontrol <- function(times, tt, surv) approx(tt, surv, xout=times)$y
tt <- 0:6

```



```

surv <- c(1, .9, .8, .75, .7, .65, .64)
formals(Scontrol) <- list(times=NULL, tt=tt, surv=surv)

# To use a mixture of two survival curves, with e.g. mixing proportions
# of .2 and .8, use the following as a guide:
#
# Scontrol <- function(times, t1, s1, t2, s2)
# .2*approx(t1, s1, xout=times)$y + .8*approx(t2, s2, xout=times)$y
# t1 <- ...; s1 <- ...; t2 <- ...; s2 <- ...;
# formals(Scontrol) <- list(times=NULL, t1=t1, s1=s1, t2=t2, s2=s2)

# Check that spower can detect a situation where generated censoring times
# are later than all failure times

rcens <- function(n) runif(n, 0, 7)
f <- Quantile2(scontrol=Scontrol, hratio=function(x).8, tmax=6)
cont <- function(n) f(n, what='control')
int <- function(n) f(n, what='intervention')
spower(rcontrol=cont, rinterv=int, rcens=rcens, nc=300, ni=300, nsim=20)

# Do an unstratified logrank test
library(survival)
# From SAS/STAT PROC LIFETEST manual, p. 1801
days <- c(179,256,262,256,255,224,225,287,319,264,237,156,270,257,242,
          157,249,180,226,268,378,355,319,256,171,325,325,217,255,256,
          291,323,253,206,206,237,211,229,234,209)
status <- c(1,1,1,1,1,0,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,0,
           0,rep(1,19))
treatment <- c(rep(1,10), rep(2,10), rep(1,10), rep(2,10))
sex <- Cs(F,F,M,F,M,F,F,M,M,M,F,F,M,M,M,F,M,F,F,M,
         M,M,M,M,F,M,M,F,F,F,M,M,M,F,F,M,F,F,F,F)
data.frame(days, status, treatment, sex)
table(treatment, status)
logrank(Surv(days, status), treatment) # agrees with p. 1807
# For stratified tests the picture is puzzling.
# survdiff(Surv(days,status) ~ treatment + strata(sex))$chisq
# is 7.246562, which does not agree with SAS (7.1609)
# But summary(coxph(Surv(days,status) ~ treatment + strata(sex)))
# yields 7.16 whereas summary(coxph(Surv(days,status) ~ treatment))
# yields 5.21 as the score test, not agreeing with SAS or logrank() (5.6485)

## End(Not run)

```

**Description**

spss.get invokes the read.spss function in the **foreign** package to read an SPSS file, with a default output format of "data.frame". The label function is used to attach labels to individual

variables instead of to the data frame as done by `read.spss`. By default, integer-valued variables are converted to a storage mode of integer unless `force.single=FALSE`. Date variables are converted to R Date variables. By default, underscores in names are converted to periods.

### Usage

```
spss.get(file, lowernames=FALSE, datevars = NULL,
         use.value.labels = TRUE, to.data.frame = TRUE,
         max.value.labels = Inf, force.single=TRUE,
         allow=NULL, charfactor=FALSE, reencode = NA)
```

### Arguments

<code>file</code>	input SPSS save file. May be a file on the WWW, indicated by file starting with 'http://' or 'https://'.
<code>lowernames</code>	set to TRUE to convert variable names to lower case
<code>datevars</code>	vector of variable names containing dates to be converted to R internal format
<code>use.value.labels</code>	see <a href="#">read.spss</a>
<code>to.data.frame</code>	see <a href="#">read.spss</a> ; default is TRUE for <code>spss.get</code>
<code>max.value.labels</code>	see <a href="#">read.spss</a>
<code>force.single</code>	set to FALSE to prevent integer-valued variables from being converted from storage mode double to integer
<code>allow</code>	a vector of characters allowed by R that should not be converted to periods in variable names. By default, underscores in variable names are converted to periods as with R before version 1.9.
<code>charfactor</code>	set to TRUE to change character variables to factors if they have fewer than n/2 unique values. Blanks and null strings are converted to NAs.
<code>reencode</code>	see <a href="#">read.spss</a>

### Value

a data frame or list

### Author(s)

Frank Harrell

### See Also

[read.spss](#), [cleanup.import](#), [sas.get](#)

### Examples

```
## Not run:
w <- spss.get('/tmp/my.sav', datevars=c('birthdate', 'deathdate'))

## End(Not run)
```

---

src

*Source a File from the Current Working Directory*

---

### **Description**

src concatenates ".s" to its argument, quotes the result, and sources in the file. It sets options(last.source) to this file name so that src() can be issued to re-source the file when it is edited.

### **Usage**

src(x)

### **Arguments**

x                    an unquoted file name aside from ".s". This base file name must be a legal S name.

### **Side Effects**

Sets system option last.source

### **Author(s)**

Frank Harrell

### **See Also**

[source](#)

### **Examples**

```
## Not run:
src(myfile) # source("myfile.s")
src()      # re-source myfile.s

## End(Not run)
```

---

 stata.get

 Enhanced Importing of STATA Files
 

---

### Description

Reads a file in Stata version 5-11 binary format format into a data frame.

### Usage

```
stata.get(file, lowernames = FALSE, convert.dates = TRUE,
          convert.factors = TRUE, missing.type = FALSE,
          convert.underscore = TRUE, warn.missing.labels = TRUE,
          force.single = TRUE, allow=NULL, charfactor=FALSE, ...)
```

### Arguments

file	input SPSS save file. May be a file on the www, indicated by file starting with 'https://'.
lowernames	set to TRUE to convert variable names to lower case
convert.dates	see <a href="#">read.dta</a>
convert.factors	see <a href="#">read.dta</a>
missing.type	see <a href="#">read.dta</a>
convert.underscore	see <a href="#">read.dta</a>
warn.missing.labels	see <a href="#">read.dta</a>
force.single	set to FALSE to prevent integer-valued variables from being converted from storage mode double to integer
allow	a vector of characters allowed by R that should not be converted to periods in variable names. By default, underscores in variable names are converted to periods as with R before version 1.9.
charfactor	set to TRUE to change character variables to factors if they have fewer than n/2 unique values. Blanks and null strings are converted to NAs.
...	arguments passed to <a href="#">read.dta</a> .

### Details

stata.get invokes the [read.dta](#) function in the **foreign** package to read an STATA file, with a default output format of [data.frame](#). The [label](#) function is used to attach labels to individual variables instead of to the data frame as done by [read.dta](#). By default, integer-valued variables are converted to a storage mode of integer unless `force.single=FALSE`. Date variables are converted to R [Date](#) variables. By default, underscores in names are converted to periods.

**Value**

A data frame

**Author(s)**

Charles Dupont

**See Also**

[read.dta](#), [cleanup.import](#), [label](#), [data.frame](#), [Date](#)

**Examples**

```
## Not run:  
w <- stata.get('/tmp/my.dta')  
  
## End(Not run)
```

---

stat\_plsmo

*Add a lowess smoother without confidence bands.*

---

**Description**

Automatically selects `iter=0` for lowess if `y` is binary, otherwise uses `iter=3`.

**Usage**

```
stat_plsmo(  
  mapping = NULL,  
  data = NULL,  
  geom = "smooth",  
  position = "identity",  
  n = 80,  
  fullrange = FALSE,  
  span = 2/3,  
  fun = function(x) x,  
  na.rm = FALSE,  
  show.legend = NA,  
  inherit.aes = TRUE,  
  ...  
)
```

**Arguments**

mapping, data, geom, position, show.legend, inherit.aes	see <code>ggplot2</code> documentation
n	number of points to evaluate smoother at
fullrange	should the fit span the full range of the plot, or just the data
span	see <code>f</code> argument to <code>lowess</code>
fun	a function to transform smoothed <code>y</code>
na.rm	If <code>FALSE</code> (the default), removes missing values with a warning. If <code>TRUE</code> silently removes missing values.
...	other arguments are passed to smoothing function

**Value**

a data.frame with additional columns	
y	predicted value

**See Also**

[lowess](#) for loess smoother.

**Examples**

```
require(ggplot2)
c <- ggplot(mtcars, aes(qsec, wt))
c + stat_plsmo()
c + stat_plsmo() + geom_point()

c + stat_plsmo(span = 0.1) + geom_point()

# Smoothers for subsets
c <- ggplot(mtcars, aes(y=wt, x=mpg)) + facet_grid(. ~ cyl)
c + stat_plsmo() + geom_point()
c + stat_plsmo(fullrange = TRUE) + geom_point()

# Geoms and stats are automatically split by aesthetics that are factors
c <- ggplot(mtcars, aes(y=wt, x=mpg, colour=factor(cyl)))
c + stat_plsmo() + geom_point()
c + stat_plsmo(aes(fill = factor(cyl))) + geom_point()
c + stat_plsmo(fullrange=TRUE) + geom_point()

# Example with logistic regression
data("kyphosis", package="rpart")
qplot(Age, as.numeric(Kyphosis) - 1, data = kyphosis) + stat_plsmo()
```

---

string.bounding.box     *Determine Dimensions of Strings*

---

**Description**

This determines the number of rows and maximum number of columns of each string in a vector.

**Usage**

```
string.bounding.box(string, type = c("chars", "width"))
```

**Arguments**

string	vector of strings
type	character: whether to count characters or screen columns

**Value**

rows	vector containing the number of character rows in each string
columns	vector containing the maximum number of character columns in each string

**Author(s)**

Charles Dupont

**See Also**

[nchar](#), [stringDims](#)

**Examples**

```
a <- c("this is a single line string", "This is a\nmulti-line string")
stringDims(a)
```

---

string.break.line     *Break a String into Many Lines at Newlines*

---

**Description**

Takes a string and breaks it into separate substrings where there are newline characters.

**Usage**

```
string.break.line(string)
```

**Arguments**

string            character vector to be separated into many lines.

**Value**

Returns a list that is the same length of as the string argument.

Each list element is a character vector.

Each character vectors elements are the split lines of the corresponding element in the string argument vector.

**Author(s)**

Charles Dupont

**See Also**

[strsplit](#)

**Examples**

```
a <- c(' ', 'this is a single line string',  
      'This is a\nmulti-line string.')
```

```
b <- string.break.line(a)
```

---

stringDims

*String Dimentions*

---

**Description**

Finds the height and width of all the string in a character vector.

**Usage**

```
stringDims(string)
```

**Arguments**

string            vector of strings

**Details**

stringDims finds the number of characters in width and number of lines in height for each string in the string argument.



**Value**

height a vector of the number of lines in each string.  
width a vector with the number of character columns in the longest line.

**Author(s)**

Charles Dupont

**See Also**

[string.bounding.box](#), [nchar](#)

**Examples**

```
a <- c("this is a single line string", "This is a\nmulty line string")
stringDims(a)
```

---

 subplot

---

*Embed a new plot within an existing plot*


---

**Description**

Subplot will embed a new plot within an existing plot at the coordinates specified (in user units of the existing plot).

**Usage**

```
subplot(fun, x, y, size=c(1,1), vadj=0.5, hadj=0.5, pars=NULL)
```

**Arguments**

fun an expression or function defining the new plot to be embedded.  
x x-coordinate(s) of the new plot (in user coordinates of the existing plot).  
y y-coordinate(s) of the new plot, x and y can be specified in any of the ways understood by `xy.coords`.  
size The size of the embedded plot in inches if x and y have length 1.  
vadj vertical adjustment of the plot when y is a scalar, the default is to center vertically, 0 means place the bottom of the plot at y, 1 places the top of the plot at y.  
hadj horizontal adjustment of the plot when x is a scalar, the default is to center horizontally, 0 means place the left edge of the plot at x, and 1 means place the right edge of the plot at x.  
pars a list of parameters to be passed to par before running fun.

## Details

The coordinates  $x$  and  $y$  can be scalars or vectors of length 2. If vectors of length 2 then they determine the opposite corners of the rectangle for the embedded plot (and the parameters `size`, `vadj`, and `hadj` are all ignored).

If  $x$  and  $y$  are given as scalars then the plot position relative to the point and the size of the plot will be determined by the arguments `size`, `vadj`, and `hadj`. The default is to center a 1 inch by 1 inch plot at  $x, y$ . Setting `vadj` and `hadj` to  $(0, 0)$  will position the lower left corner of the plot at  $(x, y)$ .

The rectangle defined by  $x$ ,  $y$ , `size`, `vadj`, and `hadj` will be used as the plotting area of the new plot. Any tick marks, axis labels, main and sub titles will be outside of this rectangle.

Any graphical parameter settings that you would like to be in place before `fun` is evaluated can be specified in the `pars` argument (warning: specifying layout parameters here (`plt`, `mfrow`, etc.) may cause unexpected results).

After the function completes the graphical parameters will have been reset to what they were before calling the function (so you can continue to augment the original plot).

## Value

An invisible list with the graphical parameters that were in effect when the subplot was created. Passing this list to `par` will enable you to augment the embedded plot.

## Author(s)

Greg Snow <greg.snow@imail.org>

## See Also

[cnvrt.coords](#), [par](#), [symbols](#)

## Examples

```
# make an original plot
plot( 11:20, sample(51:60) )

# add some histograms

subplot( hist(rnorm(100)), 15, 55)
subplot( hist(runif(100),main='',xlab='',ylab=''), 11, 51, hadj=0, vadj=0)
subplot( hist(rexp(100, 1/3)), 20, 60, hadj=1, vadj=1, size=c(0.5,2) )
subplot( hist(rt(100,3)), c(12,16), c(57,59), pars=list(lwd=3,ask=FALSE) )

tmp <- rnorm(25)
qqnorm(tmp)
qqline(tmp)
tmp2 <- subplot( hist(tmp,xlab='',ylab='',main=''),
cnvrt.coords(0.1,0.9,'plt')$usr, vadj=1, hadj=0 )
abline(v=0, col='red') # wrong way to add a reference line to histogram

# right way to add a reference line to histogram
op <- par(no.readonly=TRUE)
```

```
par(tmp2)
abline(v=0, col='green')
par(op)
```

---

summarize

*Summarize Scalars or Matrices by Cross-Classification*


---

## Description

`summarize` is a fast version of `summary.formula(formula,method="cross",overall=FALSE)` for producing stratified summary statistics and storing them in a data frame for plotting (especially with `trellis xyplot` and `dotplot` and `Hmisc xYplot`). Unlike `aggregate`, `summarize` accepts a matrix as its first argument and a multi-valued `FUN` argument and `summarize` also labels the variables in the new data frame using their original names. Unlike methods based on `tapply`, `summarize` stores the values of the stratification variables using their original types, e.g., a numeric by variable will remain a numeric variable in the collapsed data frame. `summarize` also retains "label" attributes for variables. `summarize` works especially well with the `Hmisc xYplot` function for displaying multiple summaries of a single variable on each panel, such as means and upper and lower confidence limits.

`asNumericMatrix` converts a data frame into a numeric matrix, saving attributes to reverse the process by `matrix2dataframe`. It saves attributes that are commonly preserved across row subsetting (i.e., it does not save `dim`, `dimnames`, or `names` attributes).

`matrix2dataFrame` converts a numeric matrix back into a data frame if it was created by `asNumericMatrix`.

## Usage

```
summarize(X, by, FUN, ...,
          stat.name=deparse(substitute(X)),
          type=c('variables','matrix'), subset=TRUE,
          keepcolnames=FALSE)
```

```
asNumericMatrix(x)
```

```
matrix2dataFrame(x, at=attr(x, 'origAttributes'), restoreAll=TRUE)
```

## Arguments

- |                 |   |
|-----------------|---|
| <code>X</code>  | a vector or matrix capable of being operated on by the function specified as the <code>FUN</code> argument  |
| <code>by</code> | one or more stratification variables. If a single variable, <code>by</code> may be a vector, otherwise it should be a list. Using the <code>Hmisc llist</code> function instead of <code>list</code> will result in individual variable names being accessible to <code>summarize</code> . For example, you can specify <code>llist(age.group,sex)</code> or <code>llist(Age=age.group,sex)</code> . The latter gives <code>age.group</code> a new temporary name, <code>Age</code> . |

<code>FUN</code>	a function of a single vector argument, used to create the statistical summaries for <code>summarize</code> . <code>FUN</code> may compute any number of statistics.
<code>...</code>	extra arguments are passed to <code>FUN</code>
<code>stat.name</code>	the name to use when creating the main summary variable. By default, the name of the <code>X</code> argument is used. Set <code>stat.name</code> to <code>NULL</code> to suppress this name replacement.
<code>type</code>	Specify <code>type="matrix"</code> to store the summary variables (if there are more than one) in a matrix.
<code>subset</code>	a logical vector or integer vector of subscripts used to specify the subset of data to use in the analysis. The default is to use all observations in the data frame.
<code>keepcolnames</code>	by default when <code>type="matrix"</code> , the first column of the computed matrix is the name of the first argument to <code>summarize</code> . Set <code>keepcolnames=TRUE</code> to retain the name of the first column created by <code>FUN</code>
<code>x</code>	a data frame (for <code>asNumericMatrix</code> ) or a numeric matrix (for <code>matrix2dataFrame</code> ).
<code>at</code>	List containing attributes of original data frame that survive subsetting. Defaults to attribute <code>"origAttributes"</code> of the object <code>x</code> , created by the call to <code>asNumericMatrix</code>
<code>restoreAll</code>	set to <code>FALSE</code> to only restore attributes <code>label</code> , <code>units</code> , and <code>levels</code> instead of all attributes

### Value

For `summarize`, a data frame containing the `by` variables and the statistical summaries (the first of which is named the same as the `X` variable unless `stat.name` is given). If `type="matrix"`, the summaries are stored in a single variable in the data frame, and this variable is a matrix.

`asNumericMatrix` returns a numeric matrix and stores an object `origAttributes` as an attribute of the returned object, with original attributes of component variables, the `storage.mode`.

`matrix2dataFrame` returns a data frame.

### Author(s)

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>

### See Also

[label](#), [cut2](#), [llist](#), [by](#)

### Examples

```
## Not run:
s <- summarize(ap>1, llist(size=cut2(sz, g=4), bone), mean,
               stat.name='Proportion')
dotplot(Proportion ~ size | bone, data=s7)
```

```

## End(Not run)

set.seed(1)
temperature <- rnorm(300, 70, 10)
month <- sample(1:12, 300, TRUE)
year <- sample(2000:2001, 300, TRUE)
g <- function(x)c(Mean=mean(x,na.rm=TRUE),Median=median(x,na.rm=TRUE))
summarize(temperature, month, g)
mApply(temperature, month, g)

mApply(temperature, month, mean, na.rm=TRUE)
w <- summarize(temperature, month, mean, na.rm=TRUE)
library(lattice)
xyplot(temperature ~ month, data=w) # plot mean temperature by month

w <- summarize(temperature, llist(year,month),
               quantile, probs=c(.5,.25,.75), na.rm=TRUE, type='matrix')
xYplot(Cbind(temperature[,1],temperature[,-1]) ~ month | year, data=w)
mApply(temperature, llist(year,month),
       quantile, probs=c(.5,.25,.75), na.rm=TRUE)

# Compute the median and outer quartiles. The outer quartiles are
# displayed using "error bars"
set.seed(111)
dfr <- expand.grid(month=1:12, year=c(1997,1998), reps=1:100)
attach(dfr)
y <- abs(month-6.5) + 2*runif(length(month)) + year-1997
s <- summarize(y, llist(month,year), smedian.hilow, conf.int=.5)
s
mApply(y, llist(month,year), smedian.hilow, conf.int=.5)

xYplot(Cbind(y,Lower,Upper) ~ month, groups=year, data=s,
       keys='lines', method='alt')
# Can also do:
s <- summarize(y, llist(month,year), quantile, probs=c(.5,.25,.75),
               stat.name=c('y','Q1','Q3'))
xYplot(Cbind(y, Q1, Q3) ~ month, groups=year, data=s, keys='lines')
# To display means and bootstrapped nonparametric confidence intervals
# use for example:
s <- summarize(y, llist(month,year), smean.cl.boot)
xYplot(Cbind(y, Lower, Upper) ~ month | year, data=s)

# For each subject use the trapezoidal rule to compute the area under
# the (time,response) curve using the Hmisc trap.rule function
x <- cbind(time=c(1,2,4,7, 1,3,5,10),response=c(1,3,2,4, 1,3,2,4))
subject <- c(rep(1,4),rep(2,4))
trap.rule(x[1:4,1],x[1:4,2])
summarize(x, subject, function(y) trap.rule(y[,1],y[,2]))

## Not run:
# Another approach would be to properly re-shape the mm array below
# This assumes no missing cells. There are many other approaches.
# mApply will do this well while allowing for missing cells.

```

```

m <- tapply(y, list(year,month), quantile, probs=c(.25,.5,.75))
mm <- array(unlist(m), dim=c(3,2,12),
           dimnames=list(c('lower','median','upper'),c('1997','1998'),
                        as.character(1:12)))
# aggregate will help but it only allows you to compute one quantile
# at a time; see also the Hmisc mApply function
dframe <- aggregate(y, list(Year=year,Month=month), quantile, probs=.5)

# Compute expected life length by race assuming an exponential
# distribution - can also use summarize
g <- function(y) { # computations for one race group
  futime <- y[,1]; event <- y[,2]
  sum(futime)/sum(event) # assume event=1 for death, 0=alive
}
mApply(cbind(followup.time, death), race, g)

# To run mApply on a data frame:
xn <- asNumericMatrix(x)
m <- mApply(xn, race, h)
# Here assume h is a function that returns a matrix similar to x
matrix2dataFrame(m)

# Get stratified weighted means
g <- function(y) wtd.mean(y[,1],y[,2])
summarize(cbind(y, wts), llist(sex,race), g, stat.name='y')
mApply(cbind(y,wts), llist(sex,race), g)

# Compare speed of mApply vs. by for computing
d <- data.frame(sex=sample(c('female','male'),100000,TRUE),
               country=sample(letters,100000,TRUE),
               y1=runif(100000), y2=runif(100000))
g <- function(x) {
  y <- c(median(x[, 'y1'])-x[, 'y2']),
        med.sum =median(x[, 'y1']+x[, 'y2']))
  names(y) <- c('med.diff','med.sum')
  y
}

system.time(by(d, llist(sex=d$sex, country=d$country), g))
system.time({
  x <- asNumericMatrix(d)
  a <- subsAttr(d)
  m <- mApply(x, llist(sex=d$sex, country=d$country), g)
})
system.time({
  x <- asNumericMatrix(d)
  summarize(x, llist(sex=d$sex, country=d$country), g)
})

# An example where each subject has one record per diagnosis but sex of
# subject is duplicated for all the rows a subject has. Get the cross-
# classified frequencies of diagnosis (dx) by sex and plot the results

```

```

# with a dot plot

count <- rep(1,length(dx))
d <- summarize(count, llist(dx,sex), sum)
Dotplot(dx ~ count | sex, data=d)

## End(Not run)
d <- list(x=1:10, a=factor(rep(c('a','b'), 5)),
         b=structure(letters[1:10], label='label for b'),
         d=c(rep(TRUE,9), FALSE), f=pi*(1 : 10))
x <- asNumericMatrix(d)
attr(x, 'origAttributes')
matrix2dataFrame(x)

detach('dfr')

# Run summarize on a matrix to get column means
x <- c(1:19,NA)
y <- 101:120
z <- cbind(x, y)
g <- c(rep(1, 10), rep(2, 10))
summarize(z, g, colMeans, na.rm=TRUE, stat.name='x')
# Also works on an all numeric data frame
summarize(as.data.frame(z), g, colMeans, na.rm=TRUE, stat.name='x')

```

---

summary.formula

*Summarize Data for Making Tables and Plots*


---

## Description

summary.formula summarizes the variables listed in an S formula, computing descriptive statistics (including ones in a user-specified function). The summary statistics may be passed to print methods, plot methods for making annotated dot charts, and latex methods for typesetting tables using LaTeX. summary.formula has three methods for computing descriptive statistics on univariate or multivariate responses, subsetted by categories of other variables. The method of summarization is specified in the parameter method (see details below). For the response and cross methods, the statistics used to summarize the data may be specified in a very flexible way (e.g., the geometric mean, 33rd percentile, Kaplan-Meier 2-year survival estimate, mixtures of several statistics). The default summary statistic for these methods is the mean (the proportion of positive responses for a binary response variable). The cross method is useful for creating data frames which contain summary statistics that are passed to trellis as raw data (to make multi-panel dot charts, for example). The print methods use the print.char.matrix function to print boxed tables.

The right hand side of formula may contain mChoice (“multiple choice”) variables. When test=TRUE each choice is tested separately as a binary categorical response.

The plot method for method=“reverse” creates a temporary function Key in frame 0 as is done by the xYplot and Ecdf.formula functions. After plot runs, you can type Key() to put a legend in a default location, or e.g. Key(locator(1)) to draw a legend where you click the left mouse button. This key is for categorical variables, so to have the opportunity to put the key on the graph

you will probably want to use the command `plot(object, which="categorical")`. A second function `Key2` is created if continuous variables are being plotted. It is used the same as `Key`. If the `which` argument is not specified to `plot`, two pages of plots will be produced. If you don't define `par(mfrow=)` yourself, `plot.summary.formula.reverse` will try to lay out a multi-panel graph to best fit all the individual dot charts for continuous variables.

There is a subscripting method for objects created with `method="response"`. This can be used to print or plot selected variables or summary statistics where there would otherwise be too many on one page.

`cumcategory` is a utility function useful when summarizing an ordinal response variable. It converts such a variable having  $k$  levels to a matrix with  $k-1$  columns, where column  $i$  is a vector of zeros and ones indicating that the categorical response is in level  $i+1$  or greater. When the left hand side of `formula` is `cumcategory(y)`, the default fun will summarize it by computing all of the relevant cumulative proportions.

Functions `conTestkw`, `catTestchisq`, `ordTestpo` are the default statistical test functions for `summary.formula`. These defaults are: Wilcoxon-Kruskal-Wallis test for continuous variables, Pearson chi-square test for categorical variables, and the likelihood ratio chi-square test from the proportional odds model for ordinal variables. These three functions serve also as templates for the user to create her own testing functions that are self-defining in terms of how the results are printed or rendered in LaTeX, or plotted.

## Usage

```
## S3 method for class 'formula'
summary(formula, data=NULL, subset=NULL,
        na.action=NULL, fun = NULL,
        method = c("response", "reverse", "cross"),
        overall = method == "response" | method == "cross",
        continuous = 10, na.rm = TRUE, na.include = method != "reverse",
        g = 4, quant = c(0.025, 0.05, 0.125, 0.25, 0.375, 0.5, 0.625,
                        0.75, 0.875, 0.95, 0.975),
        nmin = if (method == "reverse") 100
                else 0,
        test = FALSE, conTest = conTestkw, catTest = catTestchisq,
        ordTest = ordTestpo, ...)

## S3 method for class 'summary.formula.response'
x[i, j, drop=FALSE]

## S3 method for class 'summary.formula.response'
print(x, vnames=c('labels','names'), prUnits=TRUE,
      abbreviate.dimnames=FALSE,
      prefix.width, min.colwidth, formatArgs=NULL, markdown=FALSE, ...)

## S3 method for class 'summary.formula.response'
plot(x, which = 1, vnames = c('labels','names'), xlim, xlab,
     pch = c(16, 1, 2, 17, 15, 3, 4, 5, 0), superposeStrata = TRUE,
     dotfont = 1, add = FALSE, reset.par = TRUE, main, subtitles = TRUE,
     ...)
```



```

## S3 method for class 'summary.formula.response'
latex(object, title = first.word(deparse(substitute(object))), caption,
      trios, vnames = c('labels', 'names'), prn = TRUE, prUnits = TRUE,
      rowlabel = '', cdec = 2, ncaption = TRUE, ...)

## S3 method for class 'summary.formula.reverse'
print(x, digits, prn = any(n != N), pctdig = 0,
      what=c('%', 'proportion'),
      npct = c('numerator', 'both', 'denominator', 'none'),
      exclude1 = TRUE, vnames = c('labels', 'names'), prUnits = TRUE,
      sep = '/', abbreviate.dimnames = FALSE,
      prefix.width = max(nchar(lab)), min.colwidth, formatArgs=NULL, round=NULL,
      prtest = c('P', 'stat', 'df', 'name'), prmsd = FALSE, long = FALSE,
      pdig = 3, eps = 0.001, ...)

## S3 method for class 'summary.formula.reverse'
plot(x, vnames = c('labels', 'names'), what = c('proportion', '%'),
     which = c('both', 'categorical', 'continuous'),
     xlim = if(what == 'proportion') c(0,1)
           else c(0,100),
     xlab = if(what=='proportion') 'Proportion'
           else 'Percentage',
     pch = c(16, 1, 2, 17, 15, 3, 4, 5, 0), exclude1 = TRUE,
     dotfont = 1, main,
     prtest = c('P', 'stat', 'df', 'name'), pdig = 3, eps = 0.001,
     conType = c('dot', 'bp', 'raw'), cex.means = 0.5, ...)

## S3 method for class 'summary.formula.reverse'
latex(object, title = first.word(deparse(substitute(object))), digits,
      prn = any(n != N), pctdig = 0, what=c('%', 'proportion'),
      npct = c("numerator", "both", "denominator", "slash", "none"),
      npct.size = 'scriptsize', Nsize = "scriptsize", exclude1 = TRUE,
      vnames=c("labels", "names"), prUnits = TRUE, middle.bold = FALSE,
      outer.size = "scriptsize", caption, rowlabel = "",
      insert.bottom = TRUE, dcolumn = FALSE, formatArgs=NULL, round = NULL,
      prtest = c('P', 'stat', 'df', 'name'), prmsd = FALSE,
      msdsize = NULL, long = dotchart, pdig = 3, eps = 0.001,
      auxCol = NULL, dotchart=FALSE, ...)

## S3 method for class 'summary.formula.cross'
print(x, twoway = nvar == 2, prnmiss = any(stats$Missing > 0), prn = TRUE,
      abbreviate.dimnames = FALSE, prefix.width = max(nchar(v)),
      min.colwidth, formatArgs = NULL, ...)

## S3 method for class 'summary.formula.cross'
latex(object, title = first.word(deparse(substitute(object))),
      twoway = nvar == 2, prnmiss = TRUE, prn = TRUE,

```

```

caption=attr(object, "heading"), vnames=c("labels", "names"),
rowlabel="", ...)

stratify(..., na.group = FALSE, shortlabel = TRUE)

## S3 method for class 'summary.formula.cross'
formula(x, ...)

cumcategory(y)

conTestkw(group, x)
catTestchisq(tab)
ordTestpo(group, x)

```

### Arguments

formula	An R formula with additive effects. For method="response" or "cross", the dependent variable has the usual connotation. For method="reverse", the dependent variable is what is usually thought of as an independent variable, and it is one that is used to stratify all of the right hand side variables. For method="response" (only), the formula may contain one or more invocations of the stratify function whose arguments are defined below. This causes the entire analysis to be stratified by cross-classifications of the combined list of stratification factors. This stratification will be reflected as major column groupings in the resulting table, or as more response columns for plotting. If formula has no dependent variable method="reverse" is the only legal value and so method defaults to "reverse" in this case.
x	an object created by summary.formula. For conTestkw a numeric vector, and for ordTestpo, a numeric or factor variable that can be considered ordered
y	a numeric, character, category, or factor vector for cumcategory. Is converted to a categorical variable is needed.
drop	logical. If TRUE the result is coerced to the lowest possible dimension.
data	name or number of a data frame. Default is the current frame.
subset	a logical vector or integer vector of subscripts used to specify the subset of data to use in the analysis. The default is to use all observations in the data frame.
na.action	function for handling missing data in the input data. The default is a function defined here called na.retain, which keeps all observations for processing, with missing variables or not.
fun	function for summarizing data in each cell. Default is to take the mean of each column of the possibly multivariate response variable. You can specify fun="%" to compute percentages (100 times the mean of a series of logical or binary variables). User-specified functions can also return a matrix. For example, you might compute quartiles on a bivariate response. Does not apply to method="reverse".
method	The default is "response", in which case the response variable may be multivariate and any number of statistics may be used to summarize them. Here

the responses are summarized separately for each of any number of independent variables. Continuous independent variables (see the continuous parameter below) are automatically stratified into *g* (see below) quantile groups (if you want to control the discretization for selected variables, use the `cut2` function on them). Otherwise, the data are subsetted by all levels of discrete right hand side variables. For multivariate responses, subjects are considered to be missing if any of the columns is missing.

The `method="reverse"` option is typically used to make baseline characteristic tables, for example. The single left hand side variable must be categorical (e.g., treatment), and the right hand side variables are broken down one at a time by the "dependent" variable. Continuous variables are described by three quantiles (quartiles by default) along with outer quantiles (used only for scaling x-axes when plotting quartiles; all are used when plotting box-percentile plots), and categorical ones are described by counts and percentages. If there is no left hand side variable, `summary` assumes that there is only one group in the data, so that only one column of summaries will appear. If there is no dependent variable in `formula`, `method` defaults to "reverse" automatically.

The `method="cross"` option allows for a multivariate dependent variable and for up to three independents. Continuous independent variables (those with at least continuous unique values) are automatically divided into *g* quantile groups. The independents are cross-classified, and marginal statistics may optionally be computed. The output of `summary.formula` in this case is a data frame containing the independent variable combinations (with levels of "All" corresponding to marginals) and the corresponding summary statistics in the matrix *S*. The output data frame is suitable for direct use in `trellis`. The `print` and `latex` typesetting methods for this method allows for a special two-way format if there are two right hand variables.

<code>overall</code>	For <code>method="reverse"</code> , setting <code>overall=TRUE</code> makes a new column with overall statistics for the whole sample. For <code>method="cross"</code> , <code>overall=TRUE</code> (the default) results in all marginal statistics being computed. For <code>trellis</code> displays (usually multi-panel dot plots), these marginals just form other categories. For "response", the default is <code>overall=TRUE</code> , causing a final row of global summary statistics to appear in tables and dot charts. If <code>test=TRUE</code> these marginal statistics are ignored in doing statistical tests.
<code>continuous</code>	specifies the threshold for when a variable is considered to be continuous (when there are at least continuous unique values). factor variables are always considered to be categorical no matter how many levels they have.
<code>na.rm</code>	TRUE (the default) to exclude NAs before passing data to <code>fun</code> to compute statistics, FALSE otherwise. <code>na.rm=FALSE</code> is useful if the response variable is a matrix and you do not wish to exclude a row of the matrix if any of the columns in that row are NA. <code>na.rm</code> also applies to summary statistic functions such as <code>smean.cl.normal</code> . For these <code>na.rm</code> defaults to TRUE unlike built-in functions.
<code>na.include</code>	for <code>method="response"</code> , set <code>na.include=FALSE</code> to exclude missing values from being counted as their own category when subsetting the response(s) by levels of a categorical variable. For <code>method="reverse"</code> set <code>na.include=TRUE</code> to keep missing values of categorical variables from being excluded from the table.

g	number of quantile groups to use when variables are automatically categorized with method="response" or "cross" using cut2
nmin	if fewer than nmin observations exist in a category for "response" (over all strata combined), that category will be ignored. For "reverse", for categories of the response variable in which there are less than or equal to nmin non-missing observations, the raw data are retained for later plotting in place of box plots.
test	applies if method="reverse". Set to TRUE to compute test statistics using tests specified in conTest and catTest.
conTest	a function of two arguments (grouping variable and a continuous variable) that returns a list with components P (the computed P-value), stat (the test statistic, either chi-square or F), df (degrees of freedom), testname (test name), statname (statistic name), namefun ("chisq", "fstat"), an optional component latexstat (LaTeX representation of statname), an optional component plotmathstat (for R - the plotmath representation of statname, as a character string), and an optional component note that contains a character string note about the test (e.g., "test not done because n < 5"). conTest is applied to continuous variables on the right-hand-side of the formula when method="reverse". The default uses the spearman2 function to run the Wilcoxon or Kruskal-Wallis test using the F distribution.
catTest	a function of a frequency table (an integer matrix) that returns a list with the same components as created by conTest. By default, the Pearson chi-square test is done, without continuity correction (the continuity correction would make the test conservative like the Fisher exact test).
ordTest	a function of a frequency table (an integer matrix) that returns a list with the same components as created by conTest. By default, the Proportional odds likelihood ratio test is done.
...	for summary.formula these are optional arguments for cut2 when variables are automatically categorized. For plot methods these arguments are passed to dotchart2. For Key and Key2 these arguments are passed to key, text, or mtitle. For print methods these are optional arguments to print.char.matrix. For latex methods these are passed to latex.default. One of the most important of these is file. Specifying file="" will cause LaTeX code to just be printed to standard output rather than be stored in a permanent file.
object	an object created by summary.formula
quant	vector of quantiles to use for summarizing data with method="reverse". This must be numbers between 0 and 1 inclusive and must include the numbers 0.5, 0.25, and 0.75 which are used for printing and for plotting quantile intervals. The outer quantiles are used for scaling the x-axes for such plots. Specify outer quantiles as 0 and 1 to scale the x-axes using the whole observed data ranges instead of the default (a 0.95 quantile interval). Box-percentile plots are drawn using all but the outer quantiles.
vnames	By default, tables and plots are usually labeled with variable labels (see the label and sas.get functions). To use the shorter variable names, specify vnames="name".

pch	vector of plotting characters to represent different groups, in order of group levels. For method="response" the characters correspond to levels of the stratify variable if superposeStrata=TRUE, and if no strata are used or if superposeStrata=FALSE, the pch vector corresponds to the which argument for method="response".
superposeStrata	If stratify was used, set superposeStrata=FALSE to make separate dot charts for each level of the stratification variable, for method='response'. The default is to superposition all strata on one dot chart.
dotfont	font for plotting points
reset.par	set to FALSE to suppress the restoring of the old par values in plot.summary.formula.response
abbreviate.dimnames	see print.char.matrix
prefix.width	see print.char.matrix
min.colwidth	minimum column width to use for boxes printed with print.char.matrix. The default is the maximum of the minimum column label length and the minimum length of entries in the data cells.
formatArgs	a list containing other arguments to pass to format.default such as scientific, e.g., formatArgs=list(scientific=c(-5,5)). For print.summary.formula.reverse and format.summary.formula.reverse, formatArgs applies only to statistics computed on continuous variables, not to percents, numerators, and denominators. The round argument may be preferred.
markdown	for print.summary.formula.response set to TRUE to use knitr::kable to produce the table in markdown format rather than using raw text output created by print.char.matrix
digits	number of significant digits to print. Default is to use the current value of the digits system option.
prn	set to TRUE to print the number of non-missing observations on the current (row) variable. The default is to print these only if any of the counts of non-missing values differs from the total number of non-missing values of the left-hand-side variable. For method="cross" the default is to always print N.
prnmiss	set to FALSE to suppress printing counts of missing values for "cross"
what	for method="reverse" specifies whether proportions or percentages are to be plotted
pctdig	number of digits to the right of the decimal place for printing percentages. The default is zero, so percents will be rounded to the nearest percent.
npct	specifies which counts are to be printed to the right of percentages. The default is to print the frequency (numerator of the percent) in parentheses. You can specify "both" to print both numerator and denominator, "denominator", "slash" to typeset horizontally using a forward slash, or "none".
npct.size	the size for typesetting npct information which appears after percents. The default is "scriptsize".
Nsize	When a second row of column headings is added showing sample sizes, Nsize specifies the LaTeX size for these subheadings. Default is "scriptsize".

exclude1	by default, method="reverse" objects will be printed, plotted, or typeset by removing redundant entries from percentage tables for categorical variables. For example, if you print the percent of females, you don't need to print the percent of males. To override this, set exclude1=FALSE.
prUnits	set to FALSE to suppress printing or latexing units attributes of variables, when method='reverse' or 'response'
sep	character to use to separate quantiles when printing method="reverse" tables
prtest	a vector of test statistic components to print if test=TRUE was in effect when summary.formula was called. Defaults to printing all components. Specify prtest=FALSE or prtest="none" to not print any tests. This applies to print, latex, and plot methods for method='reverse'.
round	for print.summary.formula.reverse and latex.summary.formula.reverse specify round to round the quantiles and optional mean and standard deviation to round digits after the decimal point
prmsd	set to TRUE to print mean and SD after the three quantiles, for continuous variables with method="reverse"
msdsize	defaults to NULL to use the current font size for the mean and standard deviation if prmsd is TRUE. Set to a character string to specify an alternate LaTeX font size.
long	set to TRUE to print the results for the first category on its own line, not on the same line with the variable label (for method="reverse" with print and latex methods)
pdig	number of digits to the right of the decimal place for printing P-values. Default is 3. This is passed to format.pval.
eps	P-values less than eps will be printed as < eps. See format.pval.
auxCol	an optional auxiliary column of information, right justified, to add in front of statistics typeset by latex.summary.formula.reverse. This argument is a list with a single element that has a name specifying the column heading. If this name includes a newline character, the portions of the string before and after the newline form respectively the main heading and the subheading (typically set in smaller font), respectively. See the extracolheads argument to latex.default. auxCol is filled with blanks when a variable being summarized takes up more than one row in the output. This happens with categorical variables.
twoway	for method="cross" with two right hand side variables, twoway controls whether the resulting table will be printed in enumeration format or as a two-way table (the default)
which	For method="response" specifies the sequential number or a vector of subscripts of statistics to plot. If you had any stratify variables, these are counted as if more statistics were computed. For method="reverse" specifies whether to plot results for categorical variables, continuous variables, or both (the default).
conType	For plotting method="reverse" plots for continuous variables, dot plots showing quartiles are drawn by default. Specify conType='bp' to draw box-percentile plots using all the quantiles in quant except the outermost ones. Means are

drawn with a solid dot and vertical reference lines are placed at the three quartiles. Specify `conType='raw'` to make a strip chart showing the raw data. This can only be used if the sample size for each left-hand-side group is less than or equal to `nmin`.

<code>cex.means</code>	character size for means in box-percentile plots; default is <code>.5</code>
<code>xlim</code>	vector of length two specifying x-axis limits. For <code>method="reverse"</code> , this is only used for plotting categorical variables. Limits for continuous variables are determined by the outer quantiles specified in <code>quant</code> .
<code>xlab</code>	x-axis label
<code>add</code>	set to <code>TRUE</code> to add to an existing plot
<code>main</code>	a main title. For <code>method="reverse"</code> this applies only to the plot for categorical variables.
<code>subtitles</code>	set to <code>FALSE</code> to suppress automatic subtitles
<code>caption</code>	character string containing LaTeX table captions.
<code>title</code>	name of resulting LaTeX file omitting the <code>.tex</code> suffix. Default is the name of the summary object. If <code>caption</code> is specified, <code>title</code> is also used for the table's symbolic reference label.
<code>trios</code>	If for <code>method="response"</code> you summarized the response(s) by using three quantiles, specify <code>trios=TRUE</code> or <code>trios=v</code> to group each set of three statistics into one column for latex output, using the format <code>a B c</code> , where the outer quantiles are in smaller font ( <code>scriptsize</code> ). For <code>trios=TRUE</code> , the overall column names are taken from the column names of the original data matrix. To give new column names, specify <code>trios=v</code> , where <code>v</code> is a vector of column names, of length $m/3$ , where <code>m</code> is the original number of columns of summary statistics.
<code>rowlabel</code>	see <code>latex.default</code> (under the help file <code>latex</code> )
<code>cdec</code>	number of decimal places to the right of the decimal point for latex. This value should be a scalar (which will be properly replicated), or a vector with length equal to the number of columns in the table. For "response" tables, this length does not count the column for <code>N</code> .
<code>ncaption</code>	set to <code>FALSE</code> to not have <code>latex.summary.formula.response</code> put sample sizes in captions
<code>i</code>	a vector of integers, or character strings containing variable names to subset on. Note that each row subsetted on in an <code>summary.formula.reverse</code> object subsets on all the levels that make up the corresponding variable (automatically).
<code>j</code>	a vector of integers representing column numbers
<code>middle.bold</code>	set to <code>TRUE</code> to have LaTeX use bold face for the middle quantile for <code>method="reverse"</code>
<code>outer.size</code>	the font size for outer quantiles for "reverse" tables
<code>insert.bottom</code>	set to <code>FALSE</code> to suppress inclusion of definitions placed at the bottom of LaTeX tables for <code>method="reverse"</code>
<code>dcolumn</code>	see <code>latex</code>
<code>na.group</code>	set to <code>TRUE</code> to have missing stratification variables given their own category (NA)
<code>shortlabel</code>	set to <code>FALSE</code> to include stratification variable names and equal signs in labels for strata levels

dotchart	set to TRUE to output a dotchart in the latex table being generated.
group	for conTest and ordTest, a numeric or factor variable with length the same as x
tab	for catTest, a frequency table such as that created by table()

**Value**

summary.formula returns a data frame or list depending on method. plot.summary.formula.reverse returns the number of pages of plots that were made.

**Side Effects**

plot.summary.formula.reverse creates a function Key and Key2 in frame 0 that will draw legends.

**Author(s)**

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>

**References**

Harrell FE (2007): Statistical tables and plots using S and LaTeX. Document available from <https://hbiostat.org/R/Hmisc/summary.pdf>.

**See Also**

[mChoice](#), [smean.sd](#), [summarize](#), [label](#), [strata](#), [dotchart2](#), [print.char.matrix](#), [update](#), [formula](#), [cut2](#), [l1ist](#), [format.default](#), [latex](#), [latexTranslate](#) [bpplt](#), [summaryM](#), [summary](#)

**Examples**

```
options(digits=3)
set.seed(173)
sex <- factor(sample(c("m","f"), 500, rep=TRUE))
age <- rnorm(500, 50, 5)
treatment <- factor(sample(c("Drug","Placebo"), 500, rep=TRUE))

# Generate a 3-choice variable; each of 3 variables has 5 possible levels
symp <- c('Headache','Stomach Ache','Hangnail',
          'Muscle Ache','Depressed')
symptom1 <- sample(symp, 500,TRUE)
symptom2 <- sample(symp, 500,TRUE)
symptom3 <- sample(symp, 500,TRUE)
Symptoms <- mChoice(symptom1, symptom2, symptom3, label='Primary Symptoms')
table(Symptoms)

# Note: In this example, some subjects have the same symptom checked
# multiple times; in practice these redundant selections would be NAs
```



```

# mChoice will ignore these redundant selections

#Frequency table sex*treatment, sex*Symptoms
summary(sex ~ treatment + Symptoms, fun=table)
# could also do summary(sex ~ treatment +
# mChoice(symptom1,symptom2,symptom3), fun=table)

#Compute mean age, separately by 3 variables
summary(age ~ sex + treatment + Symptoms)

f <- summary(treatment ~ age + sex + Symptoms, method="reverse", test=TRUE)
f
# trio of numbers represent 25th, 50th, 75th percentile
print(f, long=TRUE)
plot(f)
plot(f, conType='bp', prtest='P')
bpplt() # annotated example showing layout of bp plot

#Compute predicted probability from a logistic regression model
#For different stratifications compute receiver operating
#characteristic curve areas (C-indexes)
predicted <- plogis(.4*(sex=="m")+.15*(age-50))
positive.diagnosis <- ifelse(runif(500)<=predicted, 1, 0)
roc <- function(z) {
  x <- z[,1];
  y <- z[,2];
  n <- length(x);
  if(n<2)return(c(ROC=NA));
  n1 <- sum(y==1);
  c(ROC= (mean(rank(x)[y==1])-(n1+1)/2)/(n-n1) );
}
y <- cbind(predicted, positive.diagnosis)
options(digits=2)
summary(y ~ age + sex, fun=roc)

options(digits=3)
summary(y ~ age + sex, fun=roc, method="cross")

#Use stratify() to produce a table in which time intervals go down the
#page and going across 3 continuous variables are summarized using
#quartiles, and are stratified by two treatments

set.seed(1)
d <- expand.grid(visit=1:5, treat=c('A','B'), reps=1:100)
d$sysbp <- rnorm(100*5*2, 120, 10)
label(d$sysbp) <- 'Systolic BP'
d$diasbp <- rnorm(100*5*2, 80, 7)
d$diasbp[1] <- NA
d$age <- rnorm(100*5*2, 50, 12)
g <- function(y) {

```

```

N <- apply(y, 2, function(w) sum(!is.na(w)))
h <- function(x) {
  qu <- quantile(x, c(.25,.5,.75), na.rm=TRUE)
  names(qu) <- c('Q1','Q2','Q3')
  c(N=sum(!is.na(x)), qu)
}
w <- as.vector(apply(y, 2, h))
names(w) <- as.vector( outer(c('N','Q1','Q2','Q3'), dimnames(y)[[2]],
                           function(x,y) paste(y,x)))

w
}
#Use na.rm=FALSE to count NAs separately by column
s <- summary(cbind(age,sysbp,diasbp) ~ visit + stratify(treat),
             na.rm=FALSE, fun=g, data=d)
#The result is very wide. Re-do, putting treatment vertically
x <- with(d, factor(paste('Visit', visit, treat)))
summary(cbind(age,sysbp,diasbp) ~ x, na.rm=FALSE, fun=g, data=d)

#Compose LaTeX code directly
g <- function(y) {
  h <- function(x) {
    qu <- format(round(quantile(x, c(.25,.5,.75), na.rm=TRUE),1),nsmall=1)
    paste('{\\scriptsize(',sum(!is.na(x)),
          ')} \\hfill{\\scriptsize ', qu[1], '} \\textbf{', qu[2],
          '} {\\scriptsize ', qu[3],'}', sep='')
  }
  apply(y, 2, h)
}
s <- summary(cbind(age,sysbp,diasbp) ~ visit + stratify(treat),
             na.rm=FALSE, fun=g, data=d)
# latex(s, prn=FALSE)
## need option in latex to not print n
#Put treatment vertically
s <- summary(cbind(age,sysbp,diasbp) ~ x, fun=g, data=d, na.rm=FALSE)
# latex(s, prn=FALSE)

#Plot estimated mean life length (assuming an exponential distribution)
#separately by levels of 4 other variables. Repeat the analysis
#by levels of a stratification variable, drug. Automatically break
#continuous variables into tertiles.
#We are using the default, method='response'
## Not run:
life.expect <- function(y) c(Years=sum(y[,1])/sum(y[,2]))
attach(pbc)
require(survival)
S <- Surv(follow.up.time, death)
s2 <- summary(S ~ age + albumin + ascites + edema + stratify(drug),
             fun=life.expect, g=3)

#Note: You can summarize other response variables using the same
#independent variables using e.g. update(s2, response~.), or you
#can change the list of independent variables using e.g.

```

```

#update(s2, response ~.- ascites) or update(s2, .~.-ascites)
#You can also print, typeset, or plot subsets of s2, e.g.
#plot(s2[c('age','albumin'),]) or plot(s2[1:2,])

s2 # invokes print.summary.formula.response

#Plot results as a separate dot chart for each of the 3 strata levels
par(mfrow=c(2,2))
plot(s2, cex.labels=.6, xlim=c(0,40), superposeStrata=FALSE)

#Typeset table, creating s2.tex
w <- latex(s2, cdec=1)
#Typeset table but just print LaTeX code
latex(s2, file="") # useful for Sweave

#Take control of groups used for age. Compute 3 quartiles for
#both cholesterol and bilirubin (excluding observations that are missing
#on EITHER ONE)

age.groups <- cut2(age, c(45,60))
g <- function(y) apply(y, 2, quantile, c(.25,.5,.75))
y <- cbind(Chol=chol,Bili=bili)
label(y) <- 'Cholesterol and Bilirubin'
#You can give new column names that are not legal S names
#by enclosing them in quotes, e.g. 'Chol (mg/dl)'=chol

s <- summary(y ~ age.groups + ascites, fun=g)

par(mfrow=c(1,2), oma=c(3,0,3,0)) # allow outer margins for overall
for(ivar in 1:2) { # title
  isub <- (1:3)+(ivar-1)*3 # *3=number of quartiles/var.
  plot(s3, which=isub, main='',
       xlab=c('Cholesterol','Bilirubin')[ivar],
       pch=c(91,16,93)) # [, closed circle, ]
}
mtext(paste('Quartiles of', label(y)), adj=.5, outer=TRUE, cex=1.75)
#Overall (outer) title

prlatex(latex(s3, trios=TRUE))
# trios -> collapse 3 quartiles

#Summarize only bilirubin, but do it with two statistics:
#the mean and the median. Make separate tables for the two randomized
#groups and make plots for the active arm.

```

```

g <- function(y) c(Mean=mean(y), Median=median(y))

for(sub in c("D-penicillamine", "placebo")) {
  ss <- summary(bili ~ age.groups + ascites + chol, fun=g,
               subset=drug==sub)
  cat('\n',sub,'\n\n')
  print(ss)

  if(sub=='D-penicillamine') {
    par(mfrow=c(1,1))
    plot(s4, which=1:2, dotfont=c(1,-1), subtitles=FALSE, main='')
    #1=mean, 2=median      -1 font = open circle
    title(sub='Closed circle: mean; Open circle: median', adj=0)
    title(sub=sub, adj=1)
  }

  w <- latex(ss, append=TRUE, fi='my.tex',
            label=if(sub=='placebo') 's4b' else 's4a',
            caption=paste(label(bili),' {\em (' ,sub,')}'), sep='')
  #Note symbolic labels for tables for two subsets: s4a, s4b
  prlatex(w)
}

#Now consider examples in 'reverse' format, where the lone dependent
#variable tells the summary function how to stratify all the
#'independent' variables. This is typically used to make tables
#comparing baseline variables by treatment group, for example.

s5 <- summary(drug ~ bili + albumin + stage + protime + sex +
             age + spiders,
             method='reverse')
#To summarize all variables, use summary(drug ~., data=pbcc)
#To summarize all variables with no stratification, use
#summary(~a+b+c) or summary(~.,data=\dots)

options(digits=1)
print(s5, npct='both')
#npct='both' : print both numerators and denominators
plot(s5, which='categorical')
Key(locator(1)) # draw legend at mouse click
par(oma=c(3,0,0,0)) # leave outer margin at bottom
plot(s5, which='continuous')
Key2()           # draw legend at lower left corner of plot
                # oma= above makes this default key fit the page better

```

```

options(digits=3)
w <- latex(s5, npct='both', here=TRUE)
# creates s5.tex

#Turn to a different dataset and do cross-classifications on possibly
#more than one independent variable. The summary function with
#method='cross' produces a data frame containing the cross-
#classifications. This data frame is suitable for multi-panel
#trellis displays, although `summarize' works better for that.

attach(prostate)
size.quartile <- cut2(sz, g=4)
bone <- factor(bm,labels=c("no mets","bone mets"))

s7 <- summary(ap>1 ~ size.quartile + bone, method='cross')
#In this case, quartiles are the default so could have said sz + bone

options(digits=3)
print(s7, twoway=FALSE)
s7 # same as print(s7)
w <- latex(s7, here=TRUE) # Make s7.tex

library(trellis,TRUE)
invisible(ps.options(reset=TRUE))
trellis.device(postscript, file='demo2.ps')

dotplot(S ~ size.quartile|bone, data=s7, #s7 is name of summary stats
        xlab="Fraction ap>1", ylab="Quartile of Tumor Size")
#Can do this more quickly with summarize:
# s7 <- summarize(ap>1, llist(size=cut2(sz, g=4), bone), mean,
#                       stat.name='Proportion')
# dotplot(Proportion ~ size | bone, data=s7)

summary(age ~ stage, method='cross')
summary(age ~ stage, fun=quantile, method='cross')
summary(age ~ stage, fun=smean.sd, method='cross')
summary(age ~ stage, fun=smedian.hilow, method='cross')
summary(age ~ stage, fun=function(x) c(Mean=mean(x), Median=median(x)),
        method='cross')
#The next statements print real two-way tables
summary(cbind(age,ap) ~ stage + bone,
        fun=function(y) apply(y, 2, quantile, c(.25,.75)),
        method='cross')
options(digits=2)
summary(log(ap) ~ sz + bone,

```

```

fun=function(y) c(Mean=mean(y), quantile(y)),
method='cross')

#Summarize an ordered categorical response by all of the needed
#cumulative proportions
summary(cumcategory(disease.severity) ~ age + sex)

## End(Not run)

```

---

summaryM

*Summarize Mixed Data Types vs. Groups*


---

## Description

summaryM summarizes the variables listed in an S formula, computing descriptive statistics and optionally statistical tests for group differences. This function is typically used when there are multiple left-hand-side variables that are independently against by groups marked by a single right-hand-side variable. The summary statistics may be passed to print methods, plot methods for making annotated dot charts and extended box plots, and latex methods for typesetting tables using LaTeX. The html method uses `htmlTable::htmlTable` to typeset the table in html, by passing information to the latex method with `html=TRUE`. This is for use with Quarto/RMarkdown. The print methods use the `print.char.matrix` function to print boxed tables when `options(prType=)` has not been given or when `prType='plain'`. For plain tables, print calls the internal function `printsummaryM`. When `prType='latex'` the latex method is invoked, and when `prType='html'` html is rendered. In Quarto/RMarkdown, proper rendering will result even if `results='asis'` does not appear in the chunk header. When rendering in html at the console due to having `options(prType='html')` the table will be rendered in a viewer.

The plot method creates plotly graphics if `options(grType='plotly')`, otherwise base graphics are used. plotly graphics provide extra information such as which quantile is being displayed when hovering the mouse. Test statistics are displayed by hovering over the mean.

Continuous variables are described by three quantiles (quartiles by default) when printing, or by the following quantiles when plotting extended box plots using the `bpplt` function: 0.05, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 0.95. The box plots are scaled to the 0.025 and 0.975 quantiles of each continuous left-hand-side variable. Categorical variables are described by counts and percentages.

The left hand side of formula may contain `mChoice` ("multiple choice") variables. When `test=TRUE` each choice is tested separately as a binary categorical response.

The plot method for `method="reverse"` creates a temporary function `Key` as is done by the `xYplot` and `Ecdf.formula` functions. After plot runs, you can type `Key()` to put a legend in a default location, or e.g. `Key(locator(1))` to draw a legend where you click the left mouse button. This key is for categorical variables, so to have the opportunity to put the key on the graph you will probably want to use the command `plot(object, which="categorical")`. A second function `Key2` is created if continuous variables are being plotted. It is used the same as `Key`. If the `which` argument is not specified to plot, two pages of plots will be produced. If you don't define `par(mfrow=)` yourself, `plot.summaryM` will try to lay out a multi-panel graph to best fit all the individual charts for continuous variables.

**Usage**

```

summaryM(formula, groups=NULL, data=NULL, subset, na.action=na.retain,
          overall=FALSE, continuous=10, na.include=FALSE,
          quant=c(0.025, 0.05, 0.125, 0.25, 0.375, 0.5, 0.625,
                  0.75, 0.875, 0.95, 0.975),
          nmin=100, test=FALSE,
          conTest=conTestkw, catTest=catTestchisq,
          ordTest=ordTestpo)

## S3 method for class 'summaryM'
print(...)

printsummaryM(x, digits, prn = any(n != N),
              what=c('proportion', '%'), pctdig = if(what == '%') 0 else 2,
              npct = c('numerator', 'both', 'denominator', 'none'),
              exclude1 = TRUE, vnames = c('labels', 'names'), prUnits = TRUE,
              sep = '/', abbreviate.dimnames = FALSE,
              prefix.width = max(nchar(lab)), min.colwidth, formatArgs=NULL, round=NULL,
              prtest = c('P', 'stat', 'df', 'name'), prmsd = FALSE, long = FALSE,
              pdig = 3, eps = 0.001, prob = c(0.25, 0.5, 0.75), prN = FALSE, ...)

## S3 method for class 'summaryM'
plot(x, vnames = c('labels', 'names'),
     which = c('both', 'categorical', 'continuous'), vars=NULL,
     xlim = c(0,1),
     xlab = 'Proportion',
     pch = c(16, 1, 2, 17, 15, 3, 4, 5, 0), exclude1 = TRUE,
     main, ncols=2,
     prtest = c('P', 'stat', 'df', 'name'), pdig = 3, eps = 0.001,
     conType = c('bp', 'dot', 'raw'), cex.means = 0.5, cex=par('cex'),
     height='auto', width=700, ...)

## S3 method for class 'summaryM'
latex(object, title =
      first.word(deparse(substitute(object))),
      file=paste(title, 'tex', sep='.'), append=FALSE, digits,
      prn = any(n != N), what=c('proportion', '%'),
      pctdig = if(what == '%') 0 else 2,
      npct = c('numerator', 'both', 'denominator', 'slash', 'none'),
      npct.size = if(html) mspecs$html$smaller else 'scriptsize',
      Nsize = if(html) mspecs$html$smaller else 'scriptsize',
      exclude1 = TRUE,
      vnames=c("labels", "names"), prUnits = TRUE, middle.bold = FALSE,
      outer.size = if(html) mspecs$html$smaller else "scriptsize",
      caption, rowlabel = "", rowsep=html,
      insert.bottom = TRUE, dcolumn = FALSE, formatArgs=NULL, round=NULL,
      prtest = c('P', 'stat', 'df', 'name'), prmsd = FALSE,
      msdsize = if(html) function(x) x else NULL, brmsd=FALSE,

```

```

long = FALSE, pdig = 3, eps = 0.001,
auxCol = NULL, table.env=TRUE, tabenv1=FALSE, prob=c(0.25, 0.5, 0.75),
prN=FALSE, legend.bottom=FALSE, html=FALSE,
mspecs=markupSpecs, ...)

```

```

## S3 method for class 'summaryM'
html(object, ...)

```

## Arguments

formula	An S formula with additive effects. There may be several variables on the right hand side separated by "+", or the numeral 1, indicating that there is no grouping variable so that only margin summaries are produced. The right hand side variable, if present, must be a discrete variable producing a limited number of groups. On the left hand side there may be any number of variables, separated by "+", and these may be of mixed types. These variables are analyzed separately by the grouping variable.
groups	if there is more than one right-hand variable, specify groups as a character string containing the name of the variable used to produce columns of the table. The remaining right hand variables are combined to produce levels that cause separate tables or plots to be produced.
x	an object created by summaryM. For conTestkw a numeric vector, and for ordTestpo, a numeric or factor variable that can be considered ordered
data	name or number of a data frame. Default is the current frame.
subset	a logical vector or integer vector of subscripts used to specify the subset of data to use in the analysis. The default is to use all observations in the data frame.
na.action	function for handling missing data in the input data. The default is a function defined here called na.retain, which keeps all observations for processing, with missing variables or not.
overall	Setting overall=TRUE makes a new column with overall statistics for the whole sample. If test=TRUE these marginal statistics are ignored in doing statistical tests.
continuous	specifies the threshold for when a variable is considered to be continuous (when there are at least continuous unique values). factor variables are always considered to be categorical no matter how many levels they have.
na.include	Set na.include=TRUE to keep missing values of categorical variables from being excluded from the table.
nmin	For categories of the response variable in which there are less than or equal to nmin non-missing observations, the raw data are retained for later plotting in place of box plots.
test	Set to TRUE to compute test statistics using tests specified in conTest and catTest.
conTest	a function of two arguments (grouping variable and a continuous variable) that returns a list with components P (the computed P-value), stat (the test statistic, either chi-square or F), df (degrees of freedom), testname (test name), namefun ("chisq", "fstat"), statname (statistic name), an optional component latexstat (LaTeX representation of statname), an optional component



	plotmathstat (for R - the plotmath representation of statname, as a character string), and an optional component note that contains a character string note about the test (e.g., "test not done because $n < 5$ "). conTest is applied to continuous variables on the right-hand-side of the formula when method="reverse". The default uses the spearman2 function to run the Wilcoxon or Kruskal-Wallis test using the F distribution.
catTest	a function of a frequency table (an integer matrix) that returns a list with the same components as created by conTest. By default, the Pearson chi-square test is done, without continuity correction (the continuity correction would make the test conservative like the Fisher exact test).
ordTest	a function of a frequency table (an integer matrix) that returns a list with the same components as created by conTest. By default, the Proportional odds likelihood ratio test is done.
...	For Key and Key2 these arguments are passed to key, text, or mtitle. For print methods these are optional arguments to print.char.matrix. For latex methods these are passed to latex.default. For html the arguments are passed the latex.summaryM, and the arguments may not include file. For print the arguments are passed to printsummaryM or latex.summaryM depending on options(prType=).
object	an object created by summaryM
quant	vector of quantiles to use for summarizing continuous variables. These must be numbers between 0 and 1 inclusive and must include the numbers 0.5, 0.25, and 0.75 which are used for printing and for plotting quantile intervals. The outer quantiles are used for scaling the x-axes for such plots. Specify outer quantiles as 0 and 1 to scale the x-axes using the whole observed data ranges instead of the default (a 0.95 quantile interval). Box-percentile plots are drawn using all but the outer quantiles.
prob	vector of quantiles to use for summarizing continuous variables. These must be numbers between 0 and 1 inclusive and have previously been included in the quant argument of summaryM. The vector must be of length three. By default it contains 0.25, 0.5, and 0.75. Warning: specifying 0 and 1 as two of the quantiles will result in computing the minimum and maximum of the variable. As for many random variables the minimum will continue to become smaller as the sample size grows, and the maximum will continue to get larger. Thus the min and max are not recommended as summary statistics.
vnames	By default, tables and plots are usually labeled with variable labels (see the label and sas.get functions). To use the shorter variable names, specify vnames="name".
pch	vector of plotting characters to represent different groups, in order of group levels.
abbreviate.dimnames	see print.char.matrix
prefix.width	see print.char.matrix
min.colwidth	minimum column width to use for boxes printed with print.char.matrix. The default is the maximum of the minimum column label length and the minimum length of entries in the data cells.

formatArgs	a list containing other arguments to pass to <code>format.default</code> such as <code>scientific</code> , e.g., <code>formatArgs=list(scientific=c(-5,5))</code> . For <code>print.summary.formula.reverse</code> and <code>format.summary.formula.reverse</code> , <code>formatArgs</code> applies only to statistics computed on continuous variables, not to percents, numerators, and denominators. The <code>round</code> argument may be preferred.
digits	number of significant digits to print. Default is to use the current value of the <code>digits</code> system option.
what	specifies whether proportions or percentages are to be printed or LaTeX'd
pctdig	number of digits to the right of the decimal place for printing percentages or proportions. The default is zero if <code>what=' %'</code> , so percents will be rounded to the nearest percent. The default is 2 for proportions.
prn	set to <code>TRUE</code> to print the number of non-missing observations on the current (row) variable. The default is to print these only if any of the counts of non-missing values differs from the total number of non-missing values of the left-hand-side variable.
prN	set to <code>TRUE</code> to print the number of non-missing observations on rows that contain continuous variables.
npct	specifies which counts are to be printed to the right of percentages. The default is to print the frequency (numerator of the percent) in parentheses. You can specify <code>"both"</code> to print both numerator and denominator as a fraction, <code>"denominator"</code> , <code>"slash"</code> to typeset horizontally using a forward slash, or <code>"none"</code> .
npct.size	the size for typesetting <code>npct</code> information which appears after percents. The default is <code>"scriptsize"</code> .
Nsize	When a second row of column headings is added showing sample sizes, <code>Nsize</code> specifies the LaTeX size for these subheadings. Default is <code>"scriptsize"</code> .
exclude1	By default, <code>summaryM</code> objects will be printed, plotted, or typeset by removing redundant entries from percentage tables for categorical variables. For example, if you print the percent of females, you don't need to print the percent of males. To override this, set <code>exclude1=FALSE</code> .
prUnits	set to <code>FALSE</code> to suppress printing or latexing units attributes of variables, when <code>method='reverse'</code> or <code>'response'</code>
sep	character to use to separate quantiles when printing tables
prtest	a vector of test statistic components to print if <code>test=TRUE</code> was in effect when <code>summaryM</code> was called. Defaults to printing all components. Specify <code>prtest=FALSE</code> or <code>prtest="none"</code> to not print any tests. This applies to <code>print</code> , <code>latex</code> , and <code>plot</code> methods.
round	Specify <code>round</code> to round the quantiles and optional mean and standard deviation to round digits after the decimal point. Set <code>round='auto'</code> to try an automatic choice.
prmsd	set to <code>TRUE</code> to print mean and SD after the three quantiles, for continuous variables
msdsize	defaults to <code>NULL</code> to use the current font size for the mean and standard deviation if <code>prmsd</code> is <code>TRUE</code> . Set to a character string or function to specify an alternate LaTeX font size.

brmsd	set to TRUE to put the mean and standard deviation on a separate line, for html
long	set to TRUE to print the results for the first category on its own line, not on the same line with the variable label
pdig	number of digits to the right of the decimal place for printing P-values. Default is 3. This is passed to <code>format.pval</code> .
eps	P-values less than <code>eps</code> will be printed as <code>&lt; eps</code> . See <code>format.pval</code> .
auxCol	an optional auxiliary column of information, right justified, to add in front of statistics typeset by <code>latex.summaryM</code> . This argument is a list with a single element that has a name specifying the column heading. If this name includes a newline character, the portions of the string before and after the newline form respectively the main heading and the subheading (typically set in smaller font), respectively. See the <code>extracolheads</code> argument to <code>latex.default</code> . <code>auxCol</code> is filled with blanks when a variable being summarized takes up more than one row in the output. This happens with categorical variables.
table.env	set to FALSE to use <code>tabular</code> environment with no caption
tabenv1	set to TRUE in the case of stratification when you want only the first stratum's table to be in a table environment. This is useful when using <code>hyperref</code> .
which	Specifies whether to plot results for categorical variables, continuous variables, or both (the default).
vars	Subscripts (indexes) of variables to plot for <code>plotly</code> graphics. Default is to plot all variables of each type (categorical or continuous).
conType	For drawing plots for continuous variables, extended box plots (box-percentile-type plots) are drawn by default, using all quantiles in <code>quant</code> except for the outermost ones which are using for scaling the overall plot based on the non-stratified marginal distribution of the current response variable. Specify <code>conType='dot'</code> to draw dot plots showing the three quartiles instead. For extended box plots, means are drawn with a solid dot and vertical reference lines are placed at the three quartiles. Specify <code>conType='raw'</code> to make a strip chart showing the raw data. This can only be used if the sample size for each right-hand-side group is less than or equal to <code>nmin</code> .
cex.means	character size for means in box-percentile plots; default is <code>.5</code>
cex	character size for other plotted items
height,width	dimensions in pixels for the <code>plotly</code> subplot object containing all the extended box plots. If <code>height="auto"</code> , <code>plot.summaryM</code> will set <code>height</code> based on the number of continuous variables and <code>ncols</code> or for dot charts it will use <code>Hmisc::plotlyHeightDotchart</code> . At present <code>height</code> is ignored for extended box plots due to vertical spacing problem with <code>plotly</code> graphics.
xlim	vector of length two specifying x-axis limits. This is only used for plotting categorical variables. Limits for continuous variables are determined by the outer quantiles specified in <code>quant</code> .
xlab	x-axis label
main	a main title. This applies only to the plot for categorical variables.
ncols	number of columns for <code>plotly</code> graphics for extended box plots. Defaults to 2. Recommendation is for 1-2.

caption	character string containing LaTeX table captions.
title	name of resulting LaTeX file omitting the .tex suffix. Default is the name of the summary object. If caption is specified, title is also used for the table's symbolic reference label.
file	name of file to write LaTeX code to. Specifying file="" will cause LaTeX code to just be printed to standard output rather than be stored in a permanent file.
append	specify TRUE to add code to an existing file
rowlabel	see latex.default (under the help file latex)
rowsep	if html is TRUE, instructs the function to use a horizontal line to separate variables from one another. Recommended if brmsd is TRUE. Ignored for LaTeX.
middle.bold	set to TRUE to have LaTeX use bold face for the middle quantile
outer.size	the font size for outer quantiles
insert.bottom	set to FALSE to suppress inclusion of definitions placed at the bottom of LaTeX tables. You can also specify a character string containing other text that overrides the automatic text. At present such text always appears in the main caption for LaTeX.
legend.bottom	set to TRUE to separate the table caption and legend. This will place table legends at the bottom of LaTeX tables.
html	set to TRUE to typeset with html
mspecs	list defining markup syntax for various languages, defaults to Hmisc markupSpecs which the user can use as a starting point for editing
dcolumn	see latex

### Value

a list. `plot.summaryM` returns the number of pages of plots that were made if using base graphics, or `plotly` objects created by `plotly::subplot` otherwise. If both categorical and continuous variables were plotted, the returned object is a list with two named elements `Categorical` and `Continuous` each containing `plotly` objects. Otherwise a `plotly` object is returned. The `latex` method returns attributes `legend` and `nstrata`.

### Side Effects

`plot.summaryM` creates a function `Key` and `Key2` in frame 0 that will draw legends, if base graphics are being used.

### Author(s)

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>

### References

Harrell FE (2004): Statistical tables and plots using S and LaTeX. Document available from <https://hbiostat.org/R/Hmisc/summary.pdf>.

**See Also**

[mChoice](#), [label](#), [dotchart3](#), [print.char.matrix](#), [update](#), [formula](#), [format.default](#), [latex](#), [latexTranslate](#), [bpplt](#), [tabulr](#), [bpplotM](#), [summaryP](#)

**Examples**

```
options(digits=3)
set.seed(173)
sex <- factor(sample(c("m","f"), 500, rep=TRUE))
country <- factor(sample(c('US', 'Canada'), 500, rep=TRUE))
age <- rnorm(500, 50, 5)
sbp <- rnorm(500, 120, 12)
label(sbp) <- 'Systolic BP'
units(sbp) <- 'mmHg'
treatment <- factor(sample(c("Drug","Placebo"), 500, rep=TRUE))
treatment[1]
sbp[1] <- NA

# Generate a 3-choice variable; each of 3 variables has 5 possible levels
symp <- c('Headache','Stomach Ache','Hangnail',
         'Muscle Ache','Depressed')
symptom1 <- sample(symp, 500,TRUE)
symptom2 <- sample(symp, 500,TRUE)
symptom3 <- sample(symp, 500,TRUE)
Symptoms <- mChoice(symptom1, symptom2, symptom3, label='Primary Symptoms')
table(as.character(Symptoms))

# Note: In this example, some subjects have the same symptom checked
# multiple times; in practice these redundant selections would be NAs
# mChoice will ignore these redundant selections

f <- summaryM(age + sex + sbp + Symptoms ~ treatment, test=TRUE)
f
# trio of numbers represent 25th, 50th, 75th percentile
print(f, long=TRUE)
plot(f) # first specify options(grType='plotly') to use plotly
plot(f, conType='dot', prtest='P')
bpplt() # annotated example showing layout of bp plot

# Produce separate tables by country
f <- summaryM(age + sex + sbp + Symptoms ~ treatment + country,
              groups='treatment', test=TRUE)
f

## Not run:
getHdata(pbc)
s5 <- summaryM(bili + albumin + stage + protime + sex +
              age + spiders ~ drug, data=pbc)

print(s5, npct='both')
# npct='both' : print both numerators and denominators
plot(s5, which='categorical')
```

```

Key(locator(1)) # draw legend at mouse click
par(oma=c(3,0,0,0)) # leave outer margin at bottom
plot(s5, which='continuous') # see also bplotM
Key2()          # draw legend at lower left corner of plot
                # oma= above makes this default key fit the page better

options(digits=3)
w <- latex(s5, npct='both', here=TRUE, file='')

options(grType='plotly')
pbc <- upData(pbc, moveUnits = TRUE)
s <- summaryM(bili + albumin + alk.phos + copper + spiders + sex ~
              drug, data=pbc, test=TRUE)
# Render html
options(prType='html')
s # invokes print.summaryM
a <- plot(s)
a$Categorical
a$Continuous
plot(s, which='con')

## End(Not run)

```

---

summaryP

---

*Multi-way Summary of Proportions*


---

## Description

summaryP produces a tall and thin data frame containing numerators (freq) and denominators (denom) after stratifying the data by a series of variables. A special capability to group a series of related yes/no variables is included through the use of the `ynbind` function, for which the user specifies a final argument label used to label the panel created for that group of related variables.

If `options(grType='plotly')` is not in effect, the plot method for summaryP displays proportions as a multi-panel dot chart using the lattice package's `dotplot` function with a special panel function. Numerators and denominators of proportions are also included as text, in the same colors as used by an optional groups variable. The formula argument used in the `dotplot` call is constructed, but the user can easily reorder the variables by specifying formula, with elements named `val` (category levels), `var` (classification variable name), `freq` (calculated result) plus the overall cross-classification variables excluding groups. If `options(grType='plotly')` is in effect, the plot method makes an entirely different display using `Hmisc::dotchartpl` with `plotly` if `marginVal` is specified, whereby a stratification variable causes more finely stratified estimates to be shown slightly below the lines, with smaller and translucent symbols if data has been run through `addMarginal`. The marginal summaries are shown as the main estimates and the user can turn off display of the stratified estimates, or view their details with hover text.

The `ggplot` method for summaryP does not draw numerators and denominators but the chart is more compact than using the plot method with base graphics because `ggplot2` does not repeat category names the same way as lattice does. Variable names that are too long to fit in panel strips are renamed (1), (2), etc. and an attribute "fnvar" is added to the result; this attribute is a character

string defining the abbreviations, useful in a figure caption. The `ggplot2` object has labels for points plotted, used by `plotly::ggplotly` as hover text (see example).

The `latex` method produces one or more LaTeX tabulars containing a table representation of the result, with optional side-by-side display if `groups` is specified. Multiple tabulars result from the presence of non-group stratification factors.

## Usage

```
summaryP(formula, data = NULL, subset = NULL,
          na.action = na.retain, sort=TRUE,
          asna = c("unknown", "unspecified"), ...)
## S3 method for class 'summaryP'
plot(x, formula=NULL, groups=NULL,
      marginVal=NULL, marginLabel=marginVal,
      refgroup=NULL, exclude1=TRUE, xlim = c(-.05, 1.05),
      text.at=NULL, cex.values = 0.5,
      key = list(columns = length(groupslevels), x = 0.75,
                 y = -0.04, cex = 0.9,
                 col = lattice::trellis.par.get('superpose.symbol')$col,
                 corner=c(0,1)),
      outerlabels=TRUE, autoarrange=TRUE,
      col=colorspace::rainbow_hcl, ...)
## S3 method for class 'summaryP'
ggplot(data, mapping, groups=NULL, exclude1=TRUE,
        xlim=c(0, 1), col=NULL, shape=NULL, size=function(n) n ^ (1/4),
        sizerange=NULL, abblen=5, autoarrange=TRUE, addlayer=NULL,
        ..., environment)
## S3 method for class 'summaryP'
latex(object, groups=NULL, exclude1=TRUE, file='', round=3,
      size=NULL, append=TRUE, ...)
```

## Arguments

<code>formula</code>	a formula with the variables for whose levels proportions are computed on the left hand side, and major classification variables on the right. The formula need to include any variable later used as groups, as the data summarization does not distinguish between superpositioning and paneling. For the plot method, formula can provide an overall to the default formula for <code>dotplot()</code> .
<code>data</code>	an optional data frame. For <code>ggplot.summaryP</code> data is the result of <code>summaryP</code> .
<code>subset</code>	an optional subsetting expression or vector
<code>na.action</code>	function specifying how to handle NAs. The default is to keep all NAs in the analysis frame.
<code>sort</code>	set to FALSE to not sort category levels in descending order of global proportions
<code>asna</code>	character vector specifying level names to consider the same as NA. Set <code>asna=NULL</code> to not consider any.
<code>x</code>	an object produced by <code>summaryP</code>

groups	a character string containing the name of a superpositioning variable for obtaining further stratification within a horizontal line in the dot chart.
marginVal	if options( <code>grType='plotly'</code> ) is in effect and the data given to <code>summaryP</code> were run through <code>addMarginal</code> , specifies the category name that represents marginal summaries (usually "All").
marginLabel	specifies a different character string to use than the value of <code>marginVal</code> . For example, if marginal proportions were computed over all regions, one may specify <code>marginVal="All"</code> , <code>marginLabel="All Regions"</code> . <code>marginLabel</code> is only used for formatting graphical output.
refgroup	used when doing a <code>plotly</code> chart and a two-level group variable was used, resulting in the half-width confidence interval for the difference in two proportions to be shown, and the actual confidence limits and the difference added to hover text. See <code>dotchartpl</code> for more details.
exclude1	By default, <code>ggplot</code> , <code>plot</code> , and <code>latex</code> methods for <code>summaryP</code> remove redundant entries from tables for variables with only two levels. For example, if you print the proportion of females, you don't need to print the proportion of males. To override this, set <code>exclude1=FALSE</code> .
xlim	x-axis limits. Default is <code>c(0, 1)</code> .
text.at	specify to leave unused space to the right of each panel to prevent numerators and denominators from touching data points. <code>text.at</code> is the upper limit for scaling panels' x-axes but tick marks are only labeled up to <code>max(xlim)</code> .
cex.values	character size to use for plotting numerators and denominators
key	a list to pass to the <code>auto.key</code> argument of <code>dotplot</code> . To place a key above the entire chart use <code>auto.key=list(columns=2)</code> for example.
outerlabels	by default if there are two conditioning variables besides groups, the <code>latticeExtra</code> package's <code>useOuterStrips</code> function is used to put strip labels in the margins, usually resulting in a much prettier chart. Set to <code>FALSE</code> to prevent usage of <code>useOuterStrips</code> .
autoarrange	If <code>TRUE</code> , the formula is re-arranged so that if there are two conditioning (paneling) variables, the variable with the most levels is taken as the vertical condition.
col	a vector of colors to use to override defaults in <code>ggplot</code> . When options( <code>grType='plotly'</code> ), see <code>dotchartpl</code> .
shape	a vector of plotting symbols to override <code>ggplot</code> defaults
mapping, environment	not used; needed because of rules for generics
size	for <code>ggplot</code> , a function that transforms denominators into metrics used for the <code>size</code> aesthetic. Default is the fourth root function so that the area of symbols is proportional to the square root of sample size. Specify <code>NULL</code> to not vary point sizes. <code>size=sqrt</code> is a reasonable alternative. Set <code>size</code> to an integer to categorize the denominators into <code>size</code> quantile groups using <code>cut2</code> . Unless <code>size</code> is an integer, the legend for sizes uses the minimum and maximum denominators and 6-tiles using <code>quantile(..., type=1)</code> so that actually occurring sample sizes are used as labels. <code>size</code> is overridden to <code>NULL</code> if the range in denominators is less than 10 or the ratio of the maximum to the minimum is less than 1.2. For <code>latex</code> , <code>size</code> is an optional font size such as "small"



sizerange	a 2-vector specifying the range argument to the ggplot2 <code>scale_size_...</code> function, which is the range of sizes allowed for the points according to the denominator. The default is <code>sizerange=c(.7, 3.25)</code> but the lower limit is increased according to the ratio of maximum to minimum sample sizes.
abblen	labels of variables having only one level and having their name longer than <code>abblen</code> characters are abbreviated and documented in <code>fnvar</code> (described elsewhere here). The default <code>abblen=5</code> is good for labels plotted vertically. If labels are rotated using <code>theme</code> a better value would be 12.
...	used only for <code>plotly</code> graphics and these arguments are passed to <code>dotchartpl</code>
object	an object produced by <code>summaryP</code>
file	file name, defaults to writing to console
round	number of digits to the right of the decimal place for proportions
append	set to <code>FALSE</code> to start output over
addlayer	a ggplot layer to add to the plot object

### Value

`summaryP` produces a data frame of class "summaryP". The `plot` method produces a lattice object of class "trellis". The `latex` method produces an object of class "latex" with an additional attribute `ngrouplevels` specifying the number of levels of any groups variable and an attribute `nstrata` specifying the number of strata.

### Author(s)

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>

### See Also

[bplotM](#), [summaryM](#), [ynbind](#), [pBlock](#), [ggplot](#), [colorFacet](#)

### Examples

```
n <- 100
f <- function(na=FALSE) {
  x <- sample(c('N', 'Y'), n, TRUE)
  if(na) x[runif(100) < .1] <- NA
  x
}
set.seed(1)
d <- data.frame(x1=f(), x2=f(), x3=f(), x4=f(), x5=f(), x6=f(), x7=f(TRUE),
               age=rnorm(n, 50, 10),
               race=sample(c('Asian', 'Black/AA', 'White'), n, TRUE),
               sex=sample(c('Female', 'Male'), n, TRUE),
               treat=sample(c('A', 'B'), n, TRUE),
               region=sample(c('North America', 'Europe'), n, TRUE))
```

```

d <- upData(d, labels=c(x1='MI', x2='Stroke', x3='AKI', x4='Migraines',
                      x5='Pregnant', x6='Other event', x7='MD withdrawal',
                      race='Race', sex='Sex'))
dasna <- subset(d, region=='North America')
with(dasna, table(race, treat))
s <- summaryP(race + sex + ynbind(x1, x2, x3, x4, x5, x6, x7, label='Exclusions') ~
              region + treat, data=d)
# add exclude1=FALSE below to include female category
plot(s, groups='treat')
require(ggplot2)
ggplot(s, groups='treat')

plot(s, val ~ freq | region * var, groups='treat', outerlabels=FALSE)
# Much better looking if omit outerlabels=FALSE; see output at
# https://hbiostat.org/R/Hmisc/summaryFuns.pdf
# See more examples under bplotM

## For plotly interactive graphic that does not handle variable size
## panels well:
## require(plotly)
## g <- ggplot(s, groups='treat')
## ggplotly(g, tooltip='text')

## For nice plotly interactive graphic:
## options(grType='plotly')
## s <- summaryP(race + sex + ynbind(x1, x2, x3, x4, x5, x6, x7,
##                                label='Exclusions') ~
##              treat, data=subset(d, region='Europe'))
##
## plot(s, groups='treat', refgroup='A') # refgroup='A' does B-A differences

# Make a chart where there is a block of variables that
# are only analyzed for males. Keep redundant sex in block for demo.
# Leave extra space for numerators, denominators
sb <- summaryP(race + sex +
              pBlock(race, sex, label='Race: Males', subset=sex=='Male') ~
              region, data=d)
plot(sb, text.at=1.3)
plot(sb, groups='region', layout=c(1,3), key=list(space='top'),
      text.at=1.15)
ggplot(sb, groups='region')
## Not run:
plot(s, groups='treat')
# plot(s, groups='treat', outerlabels=FALSE) for standard lattice output
plot(s, groups='region', key=list(columns=2, space='bottom'))
require(ggplot2)
colorFacet(ggplot(s))

plot(summaryP(race + sex ~ region, data=d), exclude1=FALSE, col='green')

require(lattice)
# Make your own plot using data frame created by summaryP

```

```

useOuterStrips(dotplot(val ~ freq | region * var, groups=treat, data=s,
  xlim=c(0,1), scales=list(y='free', rot=0), xlab='Fraction',
  panel=function(x, y, subscripts, ...) {
    denom <- s$denom[subscripts]
    x <- x / denom
    panel.dotplot(x=x, y=y, subscripts=subscripts, ...) }))

# Show marginal summary for all regions combined
s <- summaryP(race + sex ~ region, data=addMarginal(d, region))
plot(s, groups='region', key=list(space='top'), layout=c(1,2))

# Show marginal summaries for both race and sex
s <- summaryP(ynbind(x1, x2, x3, x4, label='Exclusions', sort=FALSE) ~
  race + sex, data=addMarginal(d, race, sex))
plot(s, val ~ freq | sex*race)

## End(Not run)

```

summaryRc

*Graphical Summarization of Continuous Variables Against a Response*

## Description

summaryRc is a continuous version of [summary.formula](#) with method='response'. It uses the [plsmo](#) function to compute the possibly stratified [lowess](#) nonparametric regression estimates, and plots them along with the data density, with selected quantiles of the overall distribution (over strata) of each x shown as arrows on top of the graph. All the x variables must be numeric and continuous or nearly continuous.

## Usage

```

summaryRc(formula, data=NULL, subset=NULL,
  na.action=NULL, fun = function(x) x,
  na.rm = TRUE, ylab=NULL, ylim=NULL, xlim=NULL,
  nloc=NULL, datadensity=NULL,
  quant = c(0.05, 0.1, 0.25, 0.5, 0.75,
    0.90, 0.95), quantloc=c('top','bottom'),
  cex.quant=.6, srt.quant=0,
  bpplot = c('none', 'top', 'top outside', 'top inside', 'bottom'),
  height.bpplot=0.08,
  trim=NULL, test = FALSE, vnames = c('labels', 'names'), ...)

```

## Arguments

**formula** An R formula with additive effects. The formula may contain one or more invocations of the [stratify](#) function whose arguments are defined below. This causes the entire analysis to be stratified by cross-classifications of the combined list of stratification factors. This stratification will be reflected as separate lowess curves.

<code>data</code>	name or number of a data frame. Default is the current frame.
<code>subset</code>	a logical vector or integer vector of subscripts used to specify the subset of data to use in the analysis. The default is to use all observations in the data frame.
<code>na.action</code>	function for handling missing data in the input data. The default is a function defined here called <code>na.retain</code> , which keeps all observations for processing, with missing variables or not.
<code>fun</code>	function for transforming lowess estimates. Default is the identity function.
<code>na.rm</code>	TRUE (the default) to exclude NAs before passing data to <code>fun</code> to compute statistics, FALSE otherwise.
<code>ylab</code>	y-axis label. Default is label attribute of y variable, or its name.
<code>ylim</code>	y-axis limits. By default each graph is scaled on its own.
<code>xlim</code>	a list with elements named as the variable names appearing on the x-axis, with each element being a 2-vector specifying lower and upper limits. Any variable not appearing in the list will have its limits computed and possibly trimmed.
<code>nloc</code>	location for sample size. Specify <code>nloc=FALSE</code> to suppress, or <code>nloc=list(x=,y=)</code> where <code>x,y</code> are relative coordinates in the data window. Default position is in the largest empty space.
<code>datadensity</code>	see <a href="#">plsmo</a> . Defaults to TRUE if there is a stratify variable, FALSE otherwise.
<code>quant</code>	vector of quantiles to use for summarizing the marginal distribution of each x. This must be numbers between 0 and 1 inclusive. Use NULL to omit quantiles.
<code>quantloc</code>	specify <code>quantloc='bottom'</code> to place at the bottom of each plot rather than the default
<code>cex.quant</code>	character size for writing which quantiles are represented. Set to 0 to suppress quantile labels.
<code>srt.quant</code>	angle for text for quantile labels
<code>bpplot</code>	if not 'none' will draw extended box plot at location given by <code>bpplot</code> , and quantiles discussed above will be suppressed. Specifying <code>bpplot='top'</code> is the same as specifying <code>bpplot='top inside'</code> .
<code>height.bpplot</code>	height in inches of the horizontal extended box plot
<code>trim</code>	The default is to plot from the 10th smallest to the 10th largest x if the number of non-NAs exceeds 200, otherwise to use the entire range of x. Specify another quantile to use other limits, e.g., <code>trim=0.01</code> will use the first and last percentiles
<code>test</code>	Set to TRUE to plot test statistics (not yet implemented).
<code>vnames</code>	By default, plots are usually labeled with variable labels (see the <code>label</code> and <code>sas.get</code> functions). To use the shorter variable names, specify <code>vnames="names"</code> .
<code>...</code>	arguments passed to <a href="#">plsmo</a>

### Value

no value is returned

**Author(s)**

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>

**See Also**

[plsmo](#), [stratify](#), [label](#), [formula](#), [panel.bplot](#)

**Examples**

```
options(digits=3)
set.seed(177)
sex <- factor(sample(c("m","f"), 500, rep=TRUE))
age <- rnorm(500, 50, 5)
bp <- rnorm(500, 120, 7)
units(age) <- 'Years'; units(bp) <- 'mmHg'
label(bp) <- 'Systolic Blood Pressure'
L <- .5*(sex == 'm') + 0.1 * (age - 50)
y <- rbinom(500, 1, plogis(L))
par(mfrow=c(1,2))
summaryRc(y ~ age + bp)
# For x limits use 1st and 99th percentiles to frame extended box plots
summaryRc(y ~ age + bp, bplot='top', datadensity=FALSE, trim=.01)
summaryRc(y ~ age + bp + stratify(sex),
          label.curves=list(keys='lines'), nloc=list(x=.1, y=.05))
y2 <- rbinom(500, 1, plogis(L + .5))
Y <- cbind(y, y2)
summaryRc(Y ~ age + bp + stratify(sex),
          label.curves=list(keys='lines'), nloc=list(x=.1, y=.05))
```

---

summaryS

*Summarize Multiple Response Variables and Make Multipanel Scatter or Dot Plot*

---

**Description**

Multiple left-hand formula variables along with right-hand side conditioning variables are reshaped into a "tall and thin" data frame if `fun` is not specified. The resulting raw data can be plotted with the `plot` method using user-specified panel functions for `lattice` graphics, typically to make a scatterplot or loess smooths, or both. The `Hmisc` `panel.plsmo` function is handy in this context. Instead, if `fun` is specified, this function takes individual response variables (which may be matrices, as in `Surv` objects) and creates one or more summary statistics that will be computed while the resulting data frame is being collapsed to one row per condition. The `plot` method in this case plots a multi-panel dot chart using the `lattice` `dotplot` function if `panel` is not specified to `plot`. There is an option to print selected statistics as text on the panels. `summaryS` pays special attention to `Hmisc` variable annotations: `label`, `units`. When `panel` is specified in addition to `fun`, a special

x-y plot is made that assumes that the x-axis variable (typically time) is discrete. This is used for example to plot multiple quantile intervals as vertical lines next to the main point. A special panel function `mvarclPanel` is provided for this purpose.

The `plotp` method produces corresponding `plotly` graphics.

When `fun` is given and `panel` is omitted, and the result of `fun` is a vector of more than one statistic, the first statistic is taken as the main one. Any columns with names not in `textonly` will figure into the calculation of axis limits. Those in `textonly` will be printed right under the dot lines in the dot chart. Statistics with names in `textplot` will figure into limits, be plotted, and printed. `pch.stats` can be used to specify symbols for statistics after the first column. When `fun` computed three columns that are plotted, columns two and three are taken as confidence limits for which horizontal "error bars" are drawn. Two levels with different thicknesses are drawn if there are four plotted summary statistics beyond the first.

`mbarclPanel` is used to draw multiple vertical lines around the main points, such as a series of quantile intervals stratified by `x` and paneling variables. If `mbarclPanel` finds a column of an argument `yother` that is named "se", and if there are exactly two levels to a superpositioning variable, the half-height of the approximate 0.95 confidence interval for the difference between two point estimates is shown, positioned at the midpoint of the two point estimates at an `x` value. This assumes normality of point estimates, and the standard error of the difference is the square root of the sum of squares of the two standard errors. By positioning the intervals in this fashion, a failure of the two point estimates to touch the half-confidence interval is consistent with rejecting the null hypothesis of no difference at the 0.05 level.

`mbarclpl` is the `sfun` function corresponding to `mbarclPanel` for `plotp`, and `medvpl` is the `sfun` replacement for `medvPanel`.

`medvPanel` takes raw data and plots median `y` vs. `x`, along with confidence intervals and half-interval for the difference in medians as with `mbarclPanel`. Quantile intervals are optional. Very transparent vertical violin plots are added by default. Unlike `panel.violin`, only half of the violin is plotted, and when there are two superpose groups they are side-by-side in different colors.

For `plotp`, the function corresponding to `medvPanel` is `medvpl`, which draws back-to-back spike histograms, optional Gini mean difference, optional SD, quantiles (thin line version of box plot with 0.05 0.25 0.5 0.75 0.95 quantiles), and half-width confidence interval for differences in medians. For quantiles, the Harrell-Davis estimator is used.

## Usage

```
summaryS(formula, fun = NULL, data = NULL, subset = NULL,
         na.action = na.retain, continuous=10, ...)
```

```
## S3 method for class 'summaryS'
plot(x, formula=NULL, groups=NULL, panel=NULL,
     paneldoesgroups=FALSE, datadensity=NULL, ylab='',
     funlabel=NULL, textonly='n', textplot=NULL,
     digits=3, custom=NULL,
     xlim=NULL, ylim=NULL, cex.strip=1, cex.values=0.5, pch.stats=NULL,
     key=list(columns=length(groupslevels),
              x=.75, y=-.04, cex=.9,
              col=lattice::trellis.par.get('superpose.symbol')$col,
              corner=c(0,1)),
```

```

outerlabels=TRUE, autoarrange=TRUE, scat1d.opts=NULL, ...)

## S3 method for class 'summaryS'
plotp(data, formula=NULL, groups=NULL, sfun=NULL,
       fitter=NULL, showpts=! length(fitter), funlabel=NULL,
       digits=5, xlim=NULL, ylim=NULL,
       shareX=TRUE, shareY=FALSE, autoarrange=TRUE, ...)

mbarclPanel(x, y, subscripts, groups=NULL, yother, ...)

medvPanel(x, y, subscripts, groups=NULL, violin=TRUE, quantiles=FALSE, ...)

mbarclpl(x, y, groups=NULL, yother, yvar=NULL, maintracename='y',
         xlim=NULL, ylim=NULL, xname='x', alphaSegments=0.45, ...)

medvpl(x, y, groups=NULL, yvar=NULL, maintracename='y',
       xlim=NULL, ylim=NULL, xlab=xname, ylab=NULL, xname='x',
       zeroline=FALSE, yother=NULL, alphaSegments=0.45,
       dhistboxp.opts=NULL, ...)

```

### Arguments

formula	a formula with possibly multiple left and right-side variables separated by +. Analysis (response) variables are on the left and are typically numeric. For plot, formula is optional and overrides the default formula inferred for the reshaped data frame.
fun	an optional summarization function, e.g., <a href="#">smean.sd</a>
data	optional input data frame. For plotp is the object produced by summaryS.
subset	optional subsetting criteria
na.action	function for dealing with NAs when constructing the model data frame
continuous	minimum number of unique values for a numeric variable to have to be considered continuous
...	ignored for summaryS and mbarclPanel, passed to strip and panel for plot. Passed to the <a href="#">density</a> function by medvPanel. For plotp, are passed to plotlyM and sfun. For mbarclpl, passed to plotlyM.
x	an object created by summaryS. For mbarclPanel is an x-axis argument provided by lattice
groups	a character string or factor specifying that one of the conditioning variables is used for superpositioning and not paneling
panel	optional lattice panel function
paneldoesgroups	set to TRUE if, like <a href="#">panel.plsmo</a> , the paneling function internally handles superpositioning for groups
datadensity	set to TRUE to add rug plots etc. using <a href="#">scat1d</a>
ylab	optional y-axis label

funlabel	optional axis label for when fun is given
textonly	names of statistics to print and not plot. By default, any statistic named "n" is only printed.
textplot	names of statistics to print and plot
digits	used if any statistics are printed as text (including plotly hover text), to specify the number of significant digits to render
custom	a function that customizes formatting of statistics that are printed as text. This is useful for generating plotmath notation. See the example in the tests directory.
xlim	optional x-axis limits
ylim	optional y-axis limits
cex.strip	size of strip labels
cex.values	size of statistics printed as text
pch.stats	symbols to use for statistics (not included the one one in column one) that are plotted. This is a named vector, with names exactly matching those created by fun. When a column does not have an entry in pch.stats, no point is drawn for that column.
key	lattice key specification
outerlabels	set to FALSE to not pass two-way charts through <a href="#">useOuterStrips</a>
autoarrange	set to FALSE to prevent plot from trying to optimize which conditioning variable is vertical
scat1d.opts	a list of options to specify to <a href="#">scat1d</a>
y, subscripts	provided by lattice
yother	passed to the panel function from the plot method based on multiple statistics computed
violin	controls whether violin plots are included
quantiles	controls whether quantile intervals are included
sfun	a function called by plotp.summaryS to compute and plot user-specified summary measures. Two functions for doing this are provided here: <a href="#">mbarclpl</a> , <a href="#">medvpl</a> .
fitter	a fitting function such as loess to smooth points. The smoothed values over a systematic grid will be evaluated and plotted as curves.
showpts	set to TRUE to show raw data points in addition to smoothed curves
shareX	TRUE to cause plotly to share a single x-axis when graphs are aligned vertically
shareY	TRUE to cause plotly to share a single y-axis when graphs are aligned horizontally
yvar	a character or factor variable used to stratify the analysis into multiple y-variables
maintracename	a default trace name when it can't be inferred
xname	x-axis variable name for hover text when it can't be inferred
xlab	x-axis label when it can't be inferred
alphaSegments	alpha saturation to draw line segments for plotly
dhistboxp.opts	list of options to pass to dhistboxp
zeroline	set to FALSE to suppress plotly zero line at x=0



**Value**

a data frame with added attributes for summaryS or a lattice object ready to render for plot

**Author(s)**

Frank Harrell

**See Also**

[summary](#), [summarize](#)

**Examples**

```
# See tests directory file summaryS.r for more examples, and summarySp.r
# for plotp examples
require(survival)
n <- 100
set.seed(1)
d <- data.frame(sbp=rnorm(n, 120, 10),
                dbp=rnorm(n, 80, 10),
                age=rnorm(n, 50, 10),
                days=sample(1:n, n, TRUE),
                S1=Surv(2*runif(n)), S2=Surv(runif(n)),
                race=sample(c('Asian', 'Black/AA', 'White'), n, TRUE),
                sex=sample(c('Female', 'Male'), n, TRUE),
                treat=sample(c('A', 'B'), n, TRUE),
                region=sample(c('North America', 'Europe'), n, TRUE),
                meda=sample(0:1, n, TRUE), medb=sample(0:1, n, TRUE))

d <- upData(d, labels=c(sbp='Systolic BP', dbp='Diastolic BP',
                      race='Race', sex='Sex', treat='Treatment',
                      days='Time Since Randomization',
                      S1='Hospitalization', S2='Re-Operation',
                      meda='Medication A', medb='Medication B'),
           units=c(sbp='mmHg', dbp='mmHg', age='Year', days='Days'))

s <- summaryS(age + sbp + dbp ~ days + region + treat, data=d)
# plot(s) # 3 pages
plot(s, groups='treat', datadensity=TRUE,
     scat1d.opts=list(lwd=.5, nhistSpike=0))
plot(s, groups='treat', panel=lattice::panel.loess,
     key=list(space='bottom', columns=2),
     datadensity=TRUE, scat1d.opts=list(lwd=.5))

# To make a plotly graph when the stratification variable region is not
# present, run the following (showpts adds raw data points):
# plotp(s, groups='treat', fitter=loess, showpts=TRUE)

# Make your own plot using data frame created by summaryP
# xyplot(y ~ days | yvar * region, groups=treat, data=s,
#        scales=list(y='free', rot=0))
```

```

# Use loess to estimate the probability of two different types of events as
# a function of time
s <- summaryS(meda + medb ~ days + treat + region, data=d)
pan <- function(...)
  panel.plsmo(..., type='l', label.curves=max(which.packet()) == 1,
             datadensity=TRUE)
plot(s, groups='treat', panel=pan, paneldoesgroups=TRUE,
     scat1d.opts=list(lwd=.7), cex.strip=.8)

# Repeat using intervals instead of nonparametric smoother
pan <- function(...) # really need mobs > 96 to est. proportion
  panel.plsmo(..., type='l', label.curves=max(which.packet()) == 1,
             method='intervals', mobs=5)

plot(s, groups='treat', panel=pan, paneldoesgroups=TRUE, xlim=c(0, 150))

# Demonstrate dot charts of summary statistics
s <- summaryS(age + sbp + dbp ~ region + treat, data=d, fun=mean)
plot(s)
plot(s, groups='treat', funlabel=expression(bar(X)))
# Compute parametric confidence limits for mean, and include sample
# sizes by naming a column "n"

f <- function(x) {
  x <- x[! is.na(x)]
  c(smean.cl.normal(x, na.rm=FALSE), n=length(x))
}
s <- summaryS(age + sbp + dbp ~ region + treat, data=d, fun=f)
plot(s, funlabel=expression(bar(X) %+-% t[0.975] %%% s))
plot(s, groups='treat', cex.values=.65,
     key=list(space='bottom', columns=2,
             text=c('Treatment A:', 'Treatment B:')))

# For discrete time, plot Harrell-Davis quantiles of y variables across
# time using different line characteristics to distinguish quantiles
d <- upData(d, days=round(days / 30) * 30)
g <- function(y) {
  probs <- c(0.05, 0.125, 0.25, 0.375)
  probs <- sort(c(probs, 1 - probs))
  y <- y[! is.na(y)]
  w <- hdquantile(y, probs)
  m <- hdquantile(y, 0.5, se=TRUE)
  se <- as.numeric(attr(m, 'se'))
  c(Median=as.numeric(m), w, se=se, n=length(y))
}
s <- summaryS(sbp + dbp ~ days + region, fun=g, data=d)
plot(s, panel=mbarclPanel)
plot(s, groups='region', panel=mbarclPanel, paneldoesgroups=TRUE)

# For discrete time, plot median y vs x along with CL for difference,
# using Harrell-Davis median estimator and its s.e., and use violin
# plots

```

```

s <- summaryS(sbp + dbp ~ days + region, data=d)
plot(s, groups='region', panel=medvPanel, paneldoesgroups=TRUE)

# Proportions and Wilson confidence limits, plus approx. Gaussian
# based half/width confidence limits for difference in probabilities
g <- function(y) {
  y <- y[!is.na(y)]
  n <- length(y)
  p <- mean(y)
  se <- sqrt(p * (1. - p) / n)
  structure(c(binconf(sum(y), n), se=se, n=n),
            names=c('Proportion', 'Lower', 'Upper', 'se', 'n'))
}
s <- summaryS(meda + medb ~ days + region, fun=g, data=d)
plot(s, groups='region', panel=mbarclPanel, paneldoesgroups=TRUE)

```

---

symbol.freq

*Graphic Representation of a Frequency Table*


---

### Description

This function can be used to represent contingency tables graphically. Frequency counts are represented as the heights of "thermometers" by default; you can also specify `symbol='circle'` to the function. There is an option to include marginal frequencies, which are plotted on a halved scale so as to not overwhelm the plot. If you do not ask for marginal frequencies to be plotted using `marginals=T`, `symbol.freq` will ask you to click the mouse where a reference symbol is to be drawn to assist in reading the scale of the frequencies.

label attributes, if present, are used for x- and y-axis labels. Otherwise, names of calling arguments are used.

### Usage

```

symbol.freq(x, y, symbol = c("thermometer", "circle"),
           marginals = FALSE, orig.scale = FALSE,
           inches = 0.25, width = 0.15, subset, srtx = 0, ...)

```

### Arguments

x	first variable to cross-classify
y	second variable
symbol	specify "thermometer" (the default) or "circle"
marginals	set to TRUE to add marginal frequencies (scaled by half) to the plot
orig.scale	set to TRUE when the first two arguments are numeric variables; this uses their original values for x and y coordinates)
inches	see <a href="#">symbols</a>

width	see thermometers option in symbols
subset	the usual subsetting vector
srtx	rotation angle for x-axis labels
...	other arguments to pass to symbols

**Author(s)**

Frank Harrell

**See Also**

[symbols](#)

**Examples**

```
## Not run:
getHdata(titanic)
attach(titanic)
age.tertile <- cut2(titanic$age, g=3)
symbol.freq(age.tertile, pclass, marginals=T, srtx=45)
detach(2)

## End(Not run)
```

---

sys

*Run Unix or Dos Depending on System*

---

**Description**

Runs unix or dos depending on the current operating system. For R, just runs system with optional concatenation of first two arguments which are assumed named command and text.

**Usage**

```
sys(command, text=NULL, output=TRUE)
# S-Plus: sys(\dots, minimized=FALSE)
```

**Arguments**

command	system command to execute
text	text to concatenate to system command, if any (typically options or file names or both)
output	set to FALSE to not return output of command as a character vector

**Value**

see unix or dos

**Side Effects**

executes system commands

**See Also**

[unix](#), [system](#)

---

t.test.cluster	<i>t-test for Clustered Data</i>
----------------	----------------------------------

---

**Description**

Does a 2-sample t-test for clustered data.

**Usage**

```
t.test.cluster(y, cluster, group, conf.int = 0.95)
## S3 method for class 't.test.cluster'
print(x, digits, ...)
```

**Arguments**

y	normally distributed response variable to test
cluster	cluster identifiers, e.g. subject ID
group	grouping variable with two values
conf.int	confidence coefficient to use for confidence limits
x	an object created by t.test.cluster
digits	number of significant digits to print
...	unused

**Value**

a matrix of statistics of class t.test.cluster

**Author(s)**

Frank Harrell

**References**

Donner A, Birkett N, Buck C, Am J Epi 114:906-914, 1981.  
Donner A, Klar N, J Clin Epi 49:435-439, 1996.  
Hsieh FY, Stat in Med 8:1195-1201, 1988.

**See Also**[t.test](#)**Examples**

```

set.seed(1)
y <- rnorm(800)
group <- sample(1:2, 800, TRUE)
cluster <- sample(1:40, 800, TRUE)
table(cluster, group)
t.test(y ~ group) # R only
t.test.cluster(y, cluster, group)
# Note: negate estimates of differences from t.test to
# compare with t.test.cluster

```

---

tabulr

---

*Interface to Tabular Function*


---

**Description**

[tabulr](#) is a front-end to the `tables` package's [tabular](#) function so that the user can take advantage of variable annotations used by the `Hmisc` package, particular those created by the [label](#), [units](#), and [upData](#) functions. When a variable appears in a [tabular](#) function, the variable `x` is found in the data argument or in the parent environment, and the [labelLatex](#) function is used to create a LaTeX label. By default any units of measurement are right justified in the current LaTeX tabular field using `hfill`; use `nofill` to list variables for which units are not right-justified with `hfill`. Once the label is constructed, the variable name is preceded by `Heading("LaTeX label")*x` in the formula before it is passed to [tabular](#). `noLabel` can be used to specify variables for which labels are ignored.

`tabulr` also replaces `trio` with `table_trio`, `N` with `table_N`, and `freq` with `table_freq` in the formula.

`table_trio` is a function that takes a numeric vector and computes the three quartiles and optionally the mean and standard deviation, and outputs a LaTeX-formatted character string representing the results. By default, calculated statistics are formatted with 3 digits to the left and 1 digit to the right of the decimal point. Running `table_options(left=l, right=r)` will use `l` and `r` digits instead. Other options that can be given to `table_options` are `prmsd=TRUE` to add mean +/- standard deviation to the result, `pn=TRUE` to add the sample size, `bold=TRUE` to set the median in bold face, `showfreq='all', 'low', 'high'` used by the `table_freq` function, `pctdec`, specifying the number of places to the right of the decimal point for percentages (default is zero), and `npct='both', 'numerator', 'denominator', 'none'` used by `table_formatpct` to control what appears after the percent. Option `pnformat` may be specified to control the formatting for `pn`. The default is `"(n=.)"`. Specify `pnformat="non"` to suppress `"n="`. `pnwhen` specifies when to print the number of observations. The default is `"always"`. Specify `pnwhen="ifna"` to include `n` only if there are missing values in the vector being processed.

`tabulr` substitutes `table_N` for `N` in the formula. This is used to create column headings for the number of observations, without a row label.

`table_freq` analyzes a character variable to compute, for a single output cell, the percents, numerator, and denominator for each category, or optimally just the maximum or minimum, as specified by `table_options(showfreq)`.

`table_formatpct` is a function that formats percents depending on settings of options in `table_options`.

`nFm` is a function that calls `sprintf` to format numeric values to have a specific number of digits to the left and to the right of the point.

`table_latexdefs` writes (by default) to the console a set of LaTeX definitions that can be invoked at any point thereafter in a knitr or sweave document by naming the macro, preceeded by a single slash. The `blfootnote` macro is called with a single LaTeX argument which will appear as a footnote without a number. `keytrio` invokes `blfootnote` to define the output of `table_trio` if mean and SD are not included. If mean and SD are included, use `keytriomsd`.

## Usage

```
tabulr(formula, data = NULL, nolabel=NULL, nofill=NULL, ...)
table_trio(x)
table_freq(x)
table_formatpct(num, den)
nFm(x, left, right, neg=FALSE, pad=FALSE, html=FALSE)
table_latexdefs(file='')
```

## Arguments

<code>formula</code>	a formula suitable for <code>tabular</code> except for the addition of <code>.(variable name), .n(), trio</code> .
<code>data</code>	a data frame or list. If omitted, the parent environment is assumed to contain the variables.
<code>nolabel</code>	a formula such as <code>~ x1 + x2</code> containing the list of variables for which labels are to be ignored, forcing use of the variable name
<code>nofill</code>	a formula such as <code>~ x1 + x2</code> containing the list of variables for which units of measurement are not to be right-justified in the field using the LaTeX <code>hfill</code> directive
<code>...</code>	other arguments to <code>tabular</code>
<code>x</code>	a numeric vector
<code>num</code>	a single numerator or vector of numerators
<code>den</code>	a single denominator
<code>left, right</code>	number of places to the left and right of the decimal point, respectively
<code>neg</code>	set to TRUE if negative x values are allowed, to add one more space to the left of the decimal place
<code>pad</code>	set to TRUE to replace blanks with the LaTeX tilde placeholder
<code>html</code>	set to TRUE to make pad use an HTML space character instead of a LaTeX tilde space
<code>file</code>	location of output of <code>table_latexdefs</code>

**Value**

tabulr returns an object of class "tabular"

**Author(s)**

Frank Harrell

**See Also**

[tabular](#), [label](#), [latex](#), [summaryM](#)

**Examples**

```
## Not run:
n <- 400
set.seed(1)
d <- data.frame(country=factor(sample(c('US', 'Canada', 'Mexico'), n, TRUE)),
                sex=factor(sample(c('Female', 'Male'), n, TRUE)),
                age=rnorm(n, 50, 10),
                sbp=rnorm(n, 120, 8))
d <- upData(d,
            preghx=ifelse(sex=='Female', sample(c('No', 'Yes'), n, TRUE), NA),
            labels=c(sbp='Systolic BP', age='Age', preghx='Pregnancy History'),
            units=c(sbp='mmHg', age='years'))

contents(d)
require(tables)
invisible(booktabs()) # use booktabs LaTeX style for tabular
g <- function(x) {
  x <- x[!is.na(x)]
  if(length(x) == 0) return('')
  paste(latexNumeric(nFm(mean(x), 3, 1)),
        '\hfill{\smaller[2]}(', length(x), ')', sep='')
}
tab <- tabulr((age + Heading('Females'))*(sex == 'Female')*sbp)*
          Heading()*g + (age + sbp)*Heading()*trio ~
          Heading()*country*Heading()*sex, data=d)

# Formula after interpretation by tabulr:
# (Heading('Age\hfill {\smaller[2]} years}') * age + Heading("Females")
# * (sex == "Female") * Heading('Systolic BP {\smaller[2]} mmHg}') * sbp)
# * Heading() * g + (age + sbp) * Heading() * table_trio ~ Heading()
# * country * Heading() * sex
cat('\begin{landscape}\n')
cat('\begin{minipage}{\textwidth}\n')
cat('\keytrio\n')
latex(tab)
cat('\end{minipage}\end{landscape}\n')

getHdata(pbc)
pbc <- upData(pbc, moveUnits=TRUE)
# Convert to character to prevent tabular from stratifying
for(x in c('sex', 'stage', 'spiders')) {
  pbc[[x]] <- as.character(pbc[[x]])
}
```



```

    label(pbc[[x]]) <- paste(toupper(substring(x, 1, 1)), substring(x, 2), sep='')
  }
  table_options(pn=TRUE, showfreq='all')
  tab <- tabulr((bili + albumin + protime + age) *
               Heading()*trio +
               (sex + stage + spiders)*Heading()*freq ~ drug, data=pbcc)
  latex(tab)

## End(Not run)

```

---

testCharDateTime	<i>testCharDateTime</i>
------------------	-------------------------

---

## Description

Test Character Variables for Dates and Times

## Usage

```
testCharDateTime(x, p = 0.5, m = 0, convert = FALSE, existing = FALSE)
```

## Arguments

x	input vector of any type, but interesting cases are for character x
p	minimum proportion of non-missing non-blank values of x for which the format is one of the formats described before considering x to be of that type
m	if greater than 0, a test is applied: the number of distinct illegal values of x (values containing a letter or underscore) must not exceed m, or type character will be returned. p is set to 1.0 when m > 0.
convert	set to TRUE to convert the variable under the dominant format. If all values are NA, type will be set to 'character'.
existing	set to TRUE to return a character string with the current type of variable without examining pattern matches

## Details

For a vector x, if it is already a date-time, date, or time variable, the type is returned if convert=FALSE, or a list with that type, the original vector, and numna=0 is returned. Otherwise if x is not a character vector, a type of notcharacter is returned, or a list that includes the original x and type='notcharacter'. When x is character, the main logic is applied. The default logic (when m=0) is to consider x a date-time variable when its format is YYYY-MM-DD HH:MM:SS (:SS is optional) in more than 1/2 of the non-missing observations. It is considered to be a date if its format is YYYY-MM-DD or MM/DD/YYYY or DD-MMM-YYYY in more than 1/2 of the non-missing observations (MMM=3-letter month). A time variable has the format HH:MM:SS or HH:MM. Blank values of x (after trimming) are set to NA before proceeding.

**Value**

if `convert=FALSE`, a single character string with the type of `x`: "character", "datetime", "date", "time".  
 If `convert=TRUE`, a list with components named `type`, `x` (converted to POSIXct, Date, or chron times format), and `numna`, the number of originally non-NA values of `x` that could not be converted to the predominant format. If there were any non-convertible dates/times, the returned vector is given an additional class `special.miss` and an attribute `special.miss` which is a list with original character values (codes) and observation numbers (obs). These are summarized by `describe()`.

**Author(s)**

Frank Harrell

**Examples**

```
for(conv in c(FALSE, TRUE)) {
  print(testCharDateTime(c('2023-03-11', '2023-04-11', 'a', 'b', 'c'), convert=conv))
  print(testCharDateTime(c('2023-03-11', '2023-04-11', 'a', 'b'), convert=conv))
  print(testCharDateTime(c('2023-03-11 11:12:13', '2023-04-11 11:13:14', 'a', 'b'), convert=conv))
  print(testCharDateTime(c('2023-03-11 11:12', '2023-04-11 11:13', 'a', 'b'), convert=conv))
  print(testCharDateTime(c('3/11/2023', '4/11/2023', 'a', 'b'), convert=conv))
}
x <- c(paste0('2023-03-0', 1:9), 'a', 'a', 'a', 'b')
y <- testCharDateTime(x, convert=TRUE)$x
describe(y) # note counts of special missing values a, b
```

---

tex *function for use in graphs that are used with the psfrag package in LaTeX*

---

**Description**

`tex` is a little function to save typing when including TeX commands in graphs that are used with the `psfrag` package in LaTeX to typeset any LaTeX text inside a postscript graphic. `tex` surrounds the input character string with `'\tex[options]{'}`. This is especially useful for getting Greek letters and math symbols in postscript graphs. By default `tex` returns a string with `psfrag` commands specifying that the string be centered, not rotated, and not specially enlarged or shrunk.

**Usage**

```
tex(string, lref='c', psref='c', scale=1, srt=0)
```

**Arguments**

<code>string</code>	a character string to be processed by <code>psfrag</code> in LaTeX.
<code>lref</code>	LaTeX reference point for <code>string</code> . See the <code>psfrag</code> documentation referenced below. Default is "c" for centered (this is also the default for <code>psref</code> ).
<code>psref</code>	PostScript reference point.
<code>scale</code>	scall factor, default is 1
<code>srt</code>	rotation for <code>string</code> in degrees (default is zero)

**Value**

tex returns a modified character string.

**Author(s)**

Frank Harrell  
Department of Biostatistics  
Vanderbilt University  
<fh@fharrell.com>

**References**

Grant MC, Carlisle (1998): The PSfrag System, Version 3. Full documentation is obtained by searching [www.ctan.org](http://www.ctan.org) for 'pfgguide.ps'.

**See Also**

[postscript](#), [par](#), [ps.options](#), [mgp.axis.labels](#), [pdf](#), [trellis.device](#), [setTrellis](#)

**Examples**

```
## Not run:
pdf('test.pdf')
x <- seq(0,15,length=100)
plot(x, dchisq(x, 5), xlab=tex('$x$'),
      ylab=tex('$f(x)$'), type='l')
title(tex('Density Function of the  $\chi_{5}^{2}$  Distribution'))
dev.off()
# To process this file in LaTeX do something like
#\documentclass{article}
#\usepackage[scanall]{psfrag}
#\begin{document}
#\begin{figure}
#\includegraphics{test.ps}
#\caption{This is an example}
#\end{figure}
#\end{document}

## End(Not run)
```

## Description

transace is [ace](#) packaged for easily automatically transforming all variables in a formula without a left-hand side. transace is a fast one-iteration version of [transcan](#) without imputation of NAs. The `ggplot` method makes nice transformation plots using `ggplot2`. Binary variables are automatically kept linear, and character or factor variables are automatically treated as categorical.

`areg.boot` uses [areg](#) or [avas](#) to fit additive regression models allowing all variables in the model (including the left-hand-side) to be transformed, with transformations chosen so as to optimize certain criteria. The default method uses [areg](#) whose goal it is to maximize  $R^2$ . `method="avas"` explicitly tries to transform the response variable so as to stabilize the variance of the residuals. All-variables-transformed models tend to inflate  $R^2$  and it can be difficult to get confidence limits for each transformation. `areg.boot` solves both of these problems using the bootstrap. As with the [validate](#) function in the `rms` library, the Efron bootstrap is used to estimate the optimism in the apparent  $R^2$ , and this optimism is subtracted from the apparent  $R^2$  to obtain a bias-corrected  $R^2$ . This is done however on the transformed response variable scale.

Tests with 3 predictors show that the [avas](#) and [ace](#) estimates are unstable unless the sample size exceeds 350. Apparent  $R^2$  with low sample sizes can be very inflated, and bootstrap estimates of  $R^2$  can be even more unstable in such cases, resulting in optimism-corrected  $R^2$  that are much lower even than the actual  $R^2$ . The situation can be improved a little by restricting predictor transformations to be monotonic. On the other hand, the `areg` approach allows one to control overfitting by specifying the number of knots to use for each continuous variable in a restricted cubic spline function.

For `method="avas"` the response transformation is restricted to be monotonic. You can specify restrictions for transformations of predictors (and linearity for the response). When the first argument is a formula, the function automatically determines which variables are categorical (i.e., factor, category, or character vectors). Specify linear transformations by enclosing variables by the `identify` function (`I()`), and specify monotonicity by using `monotone(variable)`. Monotonicity restrictions are not allowed with `method="areg"`.

The [summary](#) method for `areg.boot` computes bootstrap estimates of standard errors of differences in predicted responses (usually on the original scale) for selected levels of each predictor against the lowest level of the predictor. The smearing estimator (see below) can be used here to estimate differences in predicted means, medians, or many other statistics. By default, quartiles are used for continuous predictors and all levels are used for categorical ones. See *Details* below. There is also a [plot](#) method for plotting transformation estimates, transformations for individual bootstrap re-samples, and pointwise confidence limits for transformations. Unless you already have a `par(mfrow=)` in effect with more than one row or column, `plot` will try to fit the plots on one page. A [predict](#) method computes predicted values on the original or transformed response scale, or a matrix of transformed predictors. There is a [Function](#) method for producing a list of R functions that perform the final fitted transformations. There is also a [print](#) method for `areg.boot` objects.

When estimated means (or medians or other statistical parameters) are requested for models fitted with `areg.boot` (by `summary.areg.boot` or `predict.areg.boot`), the “smearing” estimator of *Duan (1983)* is used. Here we estimate the mean of the untransformed response by computing the arithmetic mean of  $ginverse(lp + residuals)$ , where `ginverse` is the inverse of the nonparametric transformation of the response (obtained by reverse linear interpolation), `lp` is the linear predictor for an individual observation on the transformed scale, and `residuals` is the entire vector of residuals estimated from the fitted model, on the transformed scales (`n` residuals for `n` original observations). The `smearingEst` function computes the general smearing estimate. For efficiency `smearingEst`

recognizes that quantiles are transformation-preserving, i.e., when one wishes to estimate a quantile of the untransformed distribution one just needs to compute the inverse transformation of the transformed estimate after the chosen quantile of the vector of residuals is added to it. When the median is desired, the estimate is  $ginverse(lp + \text{median}(residuals))$ . See the last example for how `smearingEst` can be used outside of `areg.boot`.

`Mean` is a generic function that returns an R function to compute the estimate of the mean of a variable. Its input is typically some kind of model fit object. Likewise, `Quantile` is a generic quantile function-producing function. `Mean.areg.boot` and `Quantile.areg.boot` create functions of a vector of linear predictors that transform them into the smearing estimates of the mean or quantile of the response variable, respectively. `Quantile.areg.boot` produces exactly the same value as `predict.areg.boot` or `smearingEst`. `Mean` approximates the mapping of linear predictors to means over an evenly spaced grid of by default 200 points. Linear interpolation is used between these points. This approximate method is much faster than the full smearing estimator once `Mean` creates the function. These functions are especially useful in `nomogram` (see the example on hypothetical data).

## Usage

```
transace(formula, trim=0.01, data=environment(formula))

## S3 method for class 'transace'
print(x, ...)

## S3 method for class 'transace'
ggplot(data, mapping, ..., environment, nrow=NULL)

areg.boot(x, data, weights, subset, na.action=na.delete,
          B=100, method=c("areg", "avas"), nk=4, evaluation=100, valrsq=TRUE,
          probs=c(.25, .5, .75), tolerance=NULL)

## S3 method for class 'areg.boot'
print(x, ...)

## S3 method for class 'areg.boot'
plot(x, ylim, boot=TRUE, col.boot=2, lwd.boot=.15,
     conf.int=.95, ...)

smearingEst(transEst, inverseTrans, res,
            statistic=c('median', 'quantile', 'mean', 'fitted', 'lp'),
            q)

## S3 method for class 'areg.boot'
summary(object, conf.int=.95, values, adj.to,
        statistic='median', q, ...)

## S3 method for class 'summary.areg.boot'
print(x, ...)
```

```

## S3 method for class 'areg.boot'
predict(object, newdata,
        statistic=c("lp", "median",
                   "quantile", "mean", "fitted", "terms"),
        q=NULL, ...)

## S3 method for class 'areg.boot'
Function(object, type=c('list','individual'),
        ytype=c('transformed','inverse'),
        prefix='.', suffix='', pos=-1, ...)

Mean(object, ...)

Quantile(object, ...)

## S3 method for class 'areg.boot'
Mean(object, evaluation=200, ...)

## S3 method for class 'areg.boot'
Quantile(object, q=.5, ...)

```

## Arguments

formula	a formula without a left-hand-side variable. Variables may be enclosed in <code>monotone()</code> , <code>linear()</code> , <code>categorical()</code> to make certain assumptions about transformations. <code>categorical</code> and <code>linear</code> need not be specified if they can be summarized from the variable values.
x	for <code>areg.boot</code> x is a formula. For <code>print</code> or <code>plot</code> , an object created by <code>areg.boot</code> or <code>transace</code> . For <code>print.summary.areg.boot</code> , and object created by <code>summary.areg.boot</code> . For <code>ggplot</code> is the result of <code>transace</code> .
object	an object created by <code>areg.boot</code> , or a model fit object suitable for <code>Mean</code> or <code>Quantile</code> .
transEst	a vector of transformed values. In log-normal regression these could be predicted $\log(Y)$ for example.
inverseTrans	a function specifying the inverse transformation needed to change <code>transEst</code> to the original untransformed scale. <code>inverseTrans</code> may also be a 2-element list defining a mapping from the transformed values to untransformed values. Linear interpolation is used in this case to obtain untransform values.
trim	quantile to which to trim original and transformed values for continuous variables for purposes of plotting the transformations with <code>ggplot.transace</code>
nrow	the number of rows to graph for <code>transace</code> transformations, with the default chosen by <code>ggplot2</code>
data	data frame to use if x is a formula and variables are not already in the search list. For <code>ggplot</code> is a <code>transace</code> object.
environment, mapping	ignored

weights	a numeric vector of observation weights. By default, all observations are weighted equally.
subset	an expression to subset data if x is a formula
na.action	a function specifying how to handle NAs. Default is <code>na.delete</code> .
B	number of bootstrap samples (default=100)
method	"areg" (the default) or "avas"
nk	number of knots for continuous variables not restricted to be linear. Default is 4. One or two is not allowed. <code>nk=0</code> forces linearity for all continuous variables.
evaluation	number of equally-spaced points at which to evaluate (and save) the nonparametric transformations derived by <code>avas</code> or <code>ace</code> . Default is 100. For <code>Mean.areg.boot</code> , evaluation is the number of points at which to evaluate exact smearing estimates, to approximate them using linear interpolation (default is 200).
valrsq	set to TRUE to more quickly do bootstrapping without validating $R^2$
probs	vector probabilities denoting the quantiles of continuous predictors to use in estimating effects of those predictors
tolerance	singularity criterion; list source code for the <code>lm.fit.qr.bare</code> function.
res	a vector of residuals from the transformed model. Not required when <code>statistic="lp"</code> or <code>statistic="fitted"</code> .
statistic	statistic to estimate with the smearing estimator. For <code>smearingEst</code> , the default results in computation of the sample median of the model residuals, then <code>smearingEst</code> adds the median residual and back-transforms to get estimated median responses on the original scale. <code>statistic="lp"</code> causes predicted transformed responses to be computed. For <code>smearingEst</code> , the result (for <code>statistic="lp"</code> ) is the input argument <code>transEst</code> . <code>statistic="fitted"</code> gives predicted untransformed responses, i.e., $ginverse(lp)$ , where <code>ginverse</code> is the inverse of the estimated response transformation, estimated by reverse linear interpolation on the tabulated nonparametric response transformation or by using an explicit analytic function. <code>statistic="quantile"</code> generalizes "median" to any single quantile q which must be specified. "mean" causes the population mean response to be estimated. For <code>predict.areg.boot</code> , <code>statistic="terms"</code> returns a matrix of transformed predictors. <code>statistic</code> can also be any R function that computes a single value on a vector of values, such as <code>statistic=var</code> . Note that in this case the function name is not quoted.
q	a single quantile of the original response scale to estimate, when <code>statistic="quantile"</code> , or for <code>Quantile.areg.boot</code> .
ylim	2-vector of y-axis limits
boot	set to FALSE to not plot any bootstrapped transformations. Set it to an integer k to plot the first k bootstrap estimates.
col.boot	color for bootstrapped transformations
lwd.boot	line width for bootstrapped transformations
conf.int	confidence level (0-1) for pointwise bootstrap confidence limits and for estimated effects of predictors in <code>summary.areg.boot</code> . The latter assumes normality of the estimated effects.

values	a list of vectors of settings of the predictors, for predictors for which you want to override settings determined from probs. The list must have named components, with names corresponding to the predictors. Example: <code>values=list(x1=c(2,4,6,8), x2=c(-1,0,1))</code> specifies that <code>summary</code> is to estimate the effect on <code>y</code> of changing <code>x1</code> from 2 to 4, 2 to 6, 2 to 8, and separately, of changing <code>x2</code> from -1 to 0 and -1 to 1.
adj.to	a named vector of adjustment constants, for setting all other predictors when examining the effect of a single predictor in <code>summary</code> . The more nonlinear is the transformation of <code>y</code> the more the adjustment settings will matter. Default values are the medians of the values defined by <code>values</code> or <code>probs</code> . You only need to name the predictors for which you are overriding the default settings. Example: <code>adj.to=c(x2=0, x5=10)</code> will set <code>x2</code> to 0 and <code>x5</code> to 10 when assessing the impact of variation in the other predictors.
newdata	a data frame or list containing the same number of values of all of the predictors used in the fit. For <code>factor</code> predictors the <code>'levels'</code> attribute do not need to be in the same order as those used in the original fit, and not all levels need to be represented. If <code>newdata</code> is omitted, you can still obtain linear predictors (on the transformed response scale) and fitted values (on the original response scale), but not "terms".
type	specifies how <code>Function</code> is to return the series of functions that define the transformations of all variables. By default a list is created, with the names of the list elements being the names of the variables. Specify <code>type="individual"</code> to have separate functions created in the current environment ( <code>pos=-1</code> , the default) or in location defined by <code>pos</code> if where is specified. For the latter method, the names of the objects created are the names of the corresponding variables, prefixed by <code>prefix</code> and with <code>suffix</code> appended to the end. If any of <code>pos</code> , <code>prefix</code> , or <code>suffix</code> is specified, <code>type</code> is automatically set to <code>"individual"</code> .
ytype	By default the first function created by <code>Function</code> is the <code>y</code> -transformation. Specify <code>ytype="inverse"</code> to instead create the inverse of the transformation, to be able to obtain originally scaled <code>y</code> -values.
prefix	character string defining the prefix for function names created when <code>type="individual"</code> . By default, the function specifying the transformation for variable <code>x</code> will be named <code>.x</code> .
suffix	character string defining the suffix for the function names
pos	See <code>assign</code> .
...	arguments passed to other functions. Ignored for <code>print.transace</code> and <code>ggplot.transace</code> .

### Details

As `transace` only does one iteration over the predictors, it may not find optimal transformations and it will be dependent on the order of the predictors in `x`.

`ace` and `avas` standardize transformed variables to have mean zero and variance one for each bootstrap sample, so if a predictor is not important it will still consistently have a positive regression coefficient. Therefore using the bootstrap to estimate standard errors of the additive least squares regression coefficients would not help in drawing inferences about the importance of the predictors. To do this, `summary.areg.boot` computes estimates of, e.g., the inter-quartile range effects



of predictors in predicting the response variable (after untransforming it). As an example, at each bootstrap repetition the estimated transformed value of one of the predictors is computed at the lower quartile, median, and upper quartile of the raw value of the predictor. These transformed  $x$  values are then multiplied by the least squares estimate of the partial regression coefficient for that transformed predictor in predicting transformed  $y$ . Then these weighted transformed  $x$  values have the weighted transformed  $x$  value corresponding to the lower quartile subtracted from them, to estimate an  $x$  effect accounting for nonlinearity. The last difference computed is then the standardized effect of raising  $x$  from its lowest to its highest quartile. Before computing differences, predicted values are back-transformed to be on the original  $y$  scale in a way depending on statistic and  $q$ . The sample standard deviation of these effects (differences) is taken over the bootstrap samples, and this is used to compute approximate confidence intervals for effects and approximate  $P$ -values, both assuming normality.

`predict` does not re-insert NAs corresponding to observations that were dropped before the fit, when `newdata` is omitted.

`statistic="fitted"` estimates the same quantity as `statistic="median"` if the residuals on the transformed response have a symmetric distribution. The two provide identical estimates when the sample median of the residuals is exactly zero. The sample mean of the residuals is constrained to be exactly zero although this does not simplify anything.

## Value

`transace` returns a list of class `transace` containing these elements: `n` (number of non-missing observations used), `transformed` (a matrix containing transformed values), `rsq` (vector of  $R^2$  with which each variable can be predicted from the others), `omitted` (row numbers of data that were deleted due to NAs), `trantab` (compact transformation lookups), `levels` (original levels of character and factor variables if the input was a data frame), `trim` (value of `trim` passed to `transace`), `limits` (the limits for plotting raw and transformed variables, computed from `trim`), and `type` (a vector of transformation types used for the variables).

`areg.boot` returns a list of class `'areg.boot'` containing many elements, including (if `valrsq` is `TRUE`) `rsquare.app` and `rsquare.val`. `summary.areg.boot` returns a list of class `'summary.areg.boot'` containing a matrix of results for each predictor and a vector of adjust-to settings. It also contains the call and a `'label'` for the statistic that was computed. A `print` method for these objects handles the printing. `predict.areg.boot` returns a vector unless `statistic="terms"`, in which case it returns a matrix. `Function.areg.boot` returns by default a list of functions whose argument is one of the variables (on the original scale) and whose returned values are the corresponding transformed values. The names of the list of functions correspond to the names of the original variables. When `type="individual"`, `Function.areg.boot` invisibly returns the vector of names of the created function objects. `Mean.areg.boot` and `Quantile.areg.boot` also return functions.

`smearingEst` returns a vector of estimates of distribution parameters of class `'labelled'` so that `print.labelled` will print a label documenting the estimate that was used (see [label](#)). This label can be retrieved for other purposes by using e.g. `label(obj)`, where `obj` was the vector returned by `smearingEst`.

## Author(s)

Frank Harrell  
Department of Biostatistics

Vanderbilt University School of Medicine  
<fh@fharrell.com>

## References

Harrell FE, Lee KL, Mark DB (1996): Stat in Med 15:361–387.  
 Duan N (1983): Smearing estimate: A nonparametric retransformation method. JASA 78:605–610.  
 Wang N, Ruppert D (1995): Nonparametric estimation of the transformation in the transform-both-sides regression model. JASA 90:522–534.  
 See [avas](#), [ace](#) for primary references.

## See Also

[avas](#), [ace](#), [ols](#), [validate](#), [predab.resample](#), [label](#), [nomogram](#)

## Examples

```
# xtrans <- transace(~ monotone(age) + sex + blood.pressure + categorical(race.code))
# print(xtrans) # show R^2s and a few other things
# ggplot(xtrans) # show transformations

# Generate random data from the model y = exp(x1 + epsilon/3) where
# x1 and epsilon are Gaussian(0,1)
set.seed(171) # to be able to reproduce example
x1 <- rnorm(200)
x2 <- runif(200) # a variable that is really unrelated to y]
x3 <- factor(sample(c('cat', 'dog', 'cow'), 200, TRUE)) # also unrelated to y
y <- exp(x1 + rnorm(200)/3)
f <- areg.boot(y ~ x1 + x2 + x3, B=40)
f

plot(f)
# Note that the fitted transformation of y is very nearly log(y)
# (the appropriate one), the transformation of x1 is nearly linear,
# and the transformations of x2 and x3 are essentially flat
# (specifying monotone(x2) if method='avas' would have resulted
# in a smaller confidence band for x2)

summary(f)

# use summary(f, values=list(x2=c(.2,.5,.8))) for example if you
# want to use nice round values for judging effects

# Plot Y hat vs. Y (this doesn't work if there were NAs)
plot(fitted(f), y) # or: plot(predict(f, statistic='fitted'), y)

# Show fit of model by varying x1 on the x-axis and creating separate
# panels for x2 and x3. For x2 using only a few discrete values
```

```

newdat <- expand.grid(x1=seq(-2,2,length=100),x2=c(.25,.75),
                    x3=c('cat','dog','cow'))
yhat <- predict(f, newdat, statistic='fitted')
# statistic='mean' to get estimated mean rather than simple inverse trans.
xYplot(yhat ~ x1 | x2, groups=x3, type='l', data=newdat)

## Not run:
# Another example, on hypothetical data
f <- areg.boot(response ~ I(age) + monotone(blood.pressure) + race)
# use I(response) to not transform the response variable
plot(f, conf.int=.9)
# Check distribution of residuals
plot(fitted(f), resid(f))
qqnorm(resid(f))
# Refit this model using ols so that we can draw a nomogram of it.
# The nomogram will show the linear predictor, median, mean.
# The last two are smearing estimators.
Function(f, type='individual') # create transformation functions
f.ols <- ols(.response(response) ~ age +
            .blood.pressure(blood.pressure) + .race(race))
# Note: This model is almost exactly the same as f but there
# will be very small differences due to interpolation of
# transformations
meanr <- Mean(f) # create function of lp computing mean response
medr <- Quantile(f) # default quantile is .5
nomogram(f.ols, fun=list(Mean=meanr,Median=medr))

# Create S functions that will do the transformations
# This is a table look-up with linear interpolation
g <- Function(f)
plot(blood.pressure, g$blood.pressure(blood.pressure))
# produces the central curve in the last plot done by plot(f)

## End(Not run)

# Another simulated example, where y has a log-normal distribution
# with mean x and variance 1. Untransformed y thus has median
# exp(x) and mean exp(x + .5sigma^2) = exp(x + .5)
# First generate data from the model y = exp(x + epsilon),
# epsilon ~ Gaussian(0, 1)

set.seed(139)
n <- 1000
x <- rnorm(n)
y <- exp(x + rnorm(n))
f <- areg.boot(y ~ x, B=20)
plot(f) # note log shape for y, linear for x. Good!
xs <- c(-2, 0, 2)
d <- data.frame(x=xs)

```

```

predict(f, d, 'fitted')
predict(f, d, 'median') # almost same; median residual=-.001
exp(xs)                 # population medians
predict(f, d, 'mean')
exp(xs + .5)           # population means

# Show how smearingEst works
res <- c(-1,0,1)       # define residuals
y <- 1:5
ytrans <- log(y)
ys <- seq(.1,15,length=50)
trans.approx <- list(x=log(ys), y=ys)
options(digits=4)
smearingEst(ytrans, exp, res, 'fitted') # ignores res
smearingEst(ytrans, trans.approx, res, 'fitted') # ignores res
smearingEst(ytrans, exp, res, 'median') # median res=0
smearingEst(ytrans, exp, res+.1, 'median') # median res=.1
smearingEst(ytrans, trans.approx, res, 'median')
smearingEst(ytrans, exp, res, 'mean')
mean(exp(ytrans[2] + res)) # should equal 2nd # above
smearingEst(ytrans, trans.approx, res, 'mean')
smearingEst(ytrans, trans.approx, res, mean)
# Last argument can be any statistical function operating
# on a vector that returns a single value

```

---

transcan

*Transformations/Imputations using Canonical Variates*


---

## Description

transcan is a nonlinear additive transformation and imputation function, and there are several functions for using and operating on its results. transcan automatically transforms continuous and categorical variables to have maximum correlation with the best linear combination of the other variables. There is also an option to use a substitute criterion - maximum correlation with the first principal component of the other variables. Continuous variables are expanded as restricted cubic splines and categorical variables are expanded as contrasts (e.g., dummy variables). By default, the first canonical variate is used to find optimum linear combinations of component columns. This function is similar to [ace](#) except that transformations for continuous variables are fitted using restricted cubic splines, monotonicity restrictions are not allowed, and NAs are allowed. When a variable has any NAs, transformed scores for that variable are imputed using least squares multiple regression incorporating optimum transformations, or NAs are optionally set to constants. Shrinkage can be used to safeguard against overfitting when imputing. Optionally, imputed values on the original scale are also computed and returned. For this purpose, recursive partitioning or multinomial logistic models can optionally be used to impute categorical variables, using what is predicted to be the most probable category.

By default, transcan imputes NAs with “best guess” expected values of transformed variables, back transformed to the original scale. Values thus imputed are most like conditional medians assuming

the transformations make variables' distributions symmetric (imputed values are similar to conditional modes for categorical variables). By instead specifying `n.impute`, `transcan` does approximate multiple imputation from the distribution of each variable conditional on all other variables. This is done by sampling `n.impute` residuals from the transformed variable, with replacement (a la bootstrapping), or by default, using Rubin's approximate Bayesian bootstrap, where a sample of size `n` with replacement is selected from the residuals on `n` non-missing values of the target variable, and then a sample of size `m` with replacement is chosen from this sample, where `m` is the number of missing values needing imputation for the current multiple imputation repetition. Neither of these bootstrap procedures assume normality or even symmetry of residuals. For sometimes-missing categorical variables, optimal scores are computed by adding the "best guess" predicted mean score to random residuals off this score. Then categories having scores closest to these predicted scores are taken as the random multiple imputations (`impcat = "rpart"` is not currently allowed with `n.impute`). The literature recommends using `n.impute = 5` or greater. `transcan` provides only an approximation to multiple imputation, especially since it "freezes" the imputation model before drawing the multiple imputations rather than using different estimates of regression coefficients for each imputation. For multiple imputation, the `aregImpute` function provides a much better approximation to the full Bayesian approach while still not requiring linearity assumptions.

When you specify `n.impute` to `transcan` you can use `fit.mult.impute` to re-fit any model `n.impute` times based on `n.impute` completed datasets (if there are any sometimes missing variables not specified to `transcan`, some observations will still be dropped from these fits). After fitting `n.impute` models, `fit.mult.impute` will return the fit object from the last imputation, with coefficients replaced by the average of the `n.impute` coefficient vectors and with a component `var` equal to the imputation-corrected variance-covariance matrix using Rubin's rule. `fit.mult.impute` can also use the object created by the `mice` function in the `mice` library to draw the multiple imputations, as well as objects created by `aregImpute`. The following components of fit objects are also replaced with averages over the `n.impute` model fits: `linear.predictors`, `fitted.values`, `stats`, `means`, `icoef`, `scale`, `center`, `y.imputed`.

By specifying `fun` to `fit.mult.impute` you can run any function on the fit objects from completed datasets, with the results saved in an element named `funresults`. This facilitates running bootstrap or cross-validation separately on each completed dataset and storing all these results in a list for later processing, e.g., with the `rms` package `processMI` function. Note that for `rms`-type validation you will need to specify `fitargs=list(x=TRUE,y=TRUE)` to `fit.mult.impute` and to use special names for fun result components, such as `validate` and `calibrate` so that the result can be processed with `processMI`. When simultaneously running multiple imputation and resampling model validation you may not need values for `n.impute` or `B` (number of bootstraps) as high as usual, as the total number of repetitions will be `n.impute * B`.

`fit.mult.impute` can incorporate robust sandwich variance estimates into Rubin's rule if `robust=TRUE`.

For `ols` models fitted by `fit.mult.impute` with stacking, the  $R^2$  measure in the stacked model fit is OK, and `print.ols` computes adjusted  $R^2$  using the real sample size so it is also OK because `fit.mult.impute` corrects the stacked error degrees of freedom in the stacked fit object to reflect the real sample size.

The `summary` method for `transcan` prints the function call,  $R^2$  achieved in transforming each variable, and for each variable the coefficients of all other transformed variables that are used to estimate the transformation of the initial variable. If `imputed=TRUE` was used in the call to `transcan`, also uses the `describe` function to print a summary of imputed values. If `long = TRUE`, also prints all imputed values with observation identifiers. There is also a simple function `print.transcan` which merely prints the transformation matrix and the function call. It has an optional argument `long`, which if

set to TRUE causes detailed parameters to be printed. Instead of plotting while transcan is running, you can plot the final transformations after the fact using `plot.transcan` or `ggplot.transcan`, if the option `trantab = TRUE` was specified to transcan. If in addition the option `imputed = TRUE` was specified to transcan, `plot` and `ggplot` will show the location of imputed values (including multiples) along the axes. For `ggplot`, imputed values are shown as red plus signs.

`impute` method for transcan does imputations for a selected original data variable, on the original scale (if `imputed=TRUE` was given to transcan). If you do not specify a variable to impute, it will do imputations for all variables given to transcan which had at least one missing value. This assumes that the original variables are accessible (i.e., they have been attached) and that you want the imputed variables to have the same names as the original variables. If `n.impute` was specified to transcan you must tell `impute` which imputation to use. Results are stored in `.GlobalEnv` when `list.out` is not specified (it is recommended to use `list.out=TRUE`).

The `predict` method for transcan computes predicted variables and imputed values from a matrix of new data. This matrix should have the same column variables as the original matrix used with transcan, and in the same order (unless a formula was used with transcan).

The `Function` function is a generic function generator. `Function.transcan` creates R functions to transform variables using transformations created by transcan. These functions are useful for getting predicted values with predictors set to values on the original scale.

The `vcov` methods are defined here so that imputation-corrected variance-covariance matrices are readily extracted from `fit.mult.impute` objects, and so that `fit.mult.impute` can easily compute traditional covariance matrices for individual completed datasets.

The subscript method for transcan preserves attributes.

The `invertTabulated` function does either inverse linear interpolation or uses sampling to sample qualifying x-values having y-values near the desired values. The latter is used to get inverse values having a reasonable distribution (e.g., no floor or ceiling effects) when the transformation has a flat or nearly flat segment, resulting in a many-to-one transformation in that region. Sampling weights are a combination of the frequency of occurrence of x-values that are within `tolInverse` times the range of y and the squared distance between the associated y-values and the target y-value (`aty`).

## Usage

```
transcan(x, method=c("canonical", "pc"),
        categorical=NULL, asis=NULL, nk, imputed=FALSE, n.impute,
        boot.method=c('approximate bayesian', 'simple'),
        trantab=FALSE, transformed=FALSE,
        impcat=c("score", "multinom", "rpart"),
        mincut=40,
        inverse=c('linearInterp', 'sample'), tolInverse=.05,
        pr=TRUE, pl=TRUE, allpl=FALSE, show.na=TRUE,
        imputed.actual=c('none', 'datadensity', 'hist', 'qq', 'ecdf'),
        iter.max=50, eps=.1, curtail=TRUE,
        imp.con=FALSE, shrink=FALSE, init.cat="mode",
        nres=if(boot.method=='simple')200 else 400,
        data, subset, na.action, treeinfo=FALSE,
        rhsImp=c('mean', 'random'), details.impcat='', ...)

## S3 method for class 'transcan'
```

```

summary(object, long=FALSE, digits=6, ...)

## S3 method for class 'transcan'
print(x, long=FALSE, ...)

## S3 method for class 'transcan'
plot(x, ...)

## S3 method for class 'transcan'
ggplot(data, mapping, scale=FALSE, ..., environment)

## S3 method for class 'transcan'
impute(x, var, imputation, name, pos.in, data,
       list.out=FALSE, pr=TRUE, check=TRUE, ...)

fit.mult.impute(formula, fitter, xtrans, data, n.impute, fit.reps=FALSE,
                dtrans, derived, fun, vcovOpts=NULL,
                robust=FALSE, cluster, robmethod=c('huber', 'efron'),
                method=c('ordinary', 'stack', 'only stack'),
                funstack=TRUE, lrt=FALSE,
                pr=TRUE, subset, fitargs)

## S3 method for class 'transcan'
predict(object, newdata, iter.max=50, eps=0.01, curtail=TRUE,
        type=c("transformed", "original"),
        inverse, tolInverse, check=FALSE, ...)

Function(object, ...)

## S3 method for class 'transcan'
Function(object, prefix=".", suffix="", pos=-1, ...)

invertTabulated(x, y, freq=rep(1,length(x)),
               aty, name='value',
               inverse=c('linearInterp', 'sample'),
               tolInverse=0.05, rule=2)

## Default S3 method:
vcov(object, regcoef.only=FALSE, ...)

## S3 method for class 'fit.mult.impute'
vcov(object, regcoef.only=TRUE,
     intercepts='mid', ...)

```

## Arguments

**x** a matrix containing continuous variable values and codes for categorical variables. The matrix must have column names (`dimnames`). If row names are

present, they are used in forming the names attribute of imputed values if `imputed = TRUE`. `x` may also be a formula, in which case the model matrix is created automatically, using data in the calling frame. Advantages of using a formula are that categorical variables can be determined automatically by a variable being a `factor` variable, and variables with two unique levels are modeled `asis`. Variables with 3 unique values are considered to be categorical if a formula is specified. For a formula you may also specify that a variable is to remain untransformed by enclosing its name with the `identify` function, e.g. `I(x3)`. The user may add other variable names to the `asis` and categorical vectors. For `invertTabulated`, `x` is a vector or a list with three components: the `x` vector, the corresponding vector of transformed values, and the corresponding vector of frequencies of the pair of original and transformed variables. For `print`, `plot`, `ggplot`, `impute`, and `predict`, `x` is an object created by `transcan`.

<code>formula</code>	any R model formula
<code>fitter</code>	any R, rms, modeling function (not in quotes) that computes a vector of <code>coefficients</code> and for which <code>vcov</code> will return a variance-covariance matrix. E.g., <code>fitter = lm, glm, ols</code> . At present models involving non-regression parameters (e.g., scale parameters in parametric survival models) are not handled fully.
<code>xtrans</code>	an object created by <code>transcan</code> , <code>aregImpute</code> , or <code>mice</code>
<code>method</code>	use <code>method="canonical"</code> or any abbreviation thereof, to use canonical variates (the default). <code>method="pc"</code> transforms a variable instead so as to maximize the correlation with the first principal component of the other variables. For <code>fit.mult.impute</code> , <code>method</code> specifies whether to use standard multiple imputation (the default <code>method='ordinary'</code> ) or whether to get final coefficients from stacking all completed datasets and fitting one model. Stacking is required if likelihood ratio tests accounting for imputation are to be done. <code>method='stack'</code> means to do regular MI and stacking, which results in more valid standard errors of coefficient estimates. <code>method='only stack'</code> means that model fits are not done on individual completed datasets, and standard errors will not be very accurate.
<code>categorical</code>	a character vector of names of variables in <code>x</code> which are categorical, for which the ordering of re-scored values is not necessarily preserved. If <code>categorical</code> is omitted, it is assumed that all variables are continuous (or binary). Set <code>categorical="*"</code> to treat all variables as categorical.
<code>asis</code>	a character vector of names of variables that are not to be transformed. For these variables, the guts of <code>lm.fit method="qr"</code> is used to impute missing values. You may want to treat binary variables <code>asis</code> (this is automatic if using a formula). If <code>imputed = TRUE</code> , you may want to use <code>"categorical"</code> for binary variables if you want to force imputed values to be one of the original data values. Set <code>asis="*"</code> to treat all variables <code>asis</code> .
<code>nk</code>	number of knots to use in expanding each continuous variable (not listed in <code>asis</code> ) in a restricted cubic spline function. Default is 3 (yielding 2 parameters for a variable) if $n < 30$ , 4 if $30 \leq n < 100$ , and 5 if $n \geq 100$ (4 parameters).
<code>imputed</code>	Set to <code>TRUE</code> to return a list containing imputed values on the original scale. If the transformation for a variable is non-monotonic, imputed values are not unique. <code>transcan</code> uses the <code>approx</code> function, which returns the highest value of the variable with the transformed score equalling the imputed score. <code>imputed=TRUE</code>



also causes original-scale imputed values to be shown as tick marks on the top margin of each graph when `show.na=TRUE` (for the final iteration only). For categorical predictors, these imputed values are passed through the `jitter` function so that their frequencies can be visualized. When `n.impute` is used, each NA will have `n.impute` tick marks.

<code>n.impute</code>	number of multiple imputations. If omitted, single predicted expected value imputation is used. <code>n.impute=5</code> is frequently recommended.
<code>boot.method</code>	default is to use the approximate Bayesian bootstrap (sample with replacement from sample with replacement of the vector of residuals). You can also specify <code>boot.method="simple"</code> to use the usual bootstrap one-stage sampling with replacement.
<code>trantab</code>	Set to <code>TRUE</code> to add an attribute <code>trantab</code> to the returned matrix. This contains a vector of lists each with components <code>x</code> and <code>y</code> containing the unique values and corresponding transformed values for the columns of <code>x</code> . This is set up to be used easily with the <code>approx</code> function. You must specify <code>trantab=TRUE</code> if you want to later use the <code>predict.transcan</code> function with <code>type="original"</code> .
<code>transformed</code>	set to <code>TRUE</code> to cause <code>transcan</code> to return an object transformed containing the matrix of transformed variables
<code>impcat</code>	This argument tells how to impute categorical variables on the original scale. The default is <code>impcat="score"</code> to impute the category whose canonical variate score is closest to the predicted score. Use <code>impcat="rpart"</code> to impute categorical variables using the values of all other transformed predictors in conjunction with the <code>rpart</code> function. A better but somewhat slower approach is to use <code>impcat="multinom"</code> to fit a multinomial logistic model to the categorical variable, at the last iteration of the <code>transcan</code> algorithm. This uses the <code>multinom</code> function in the <code>nnet</code> library of the <code>MASS</code> package (which is assumed to have been installed by the user) to fit a polytomous logistic model to the current working transformations of all the other variables (using conditional mean imputation for missing predictors). Multiple imputations are made by drawing multinomial values from the vector of predicted probabilities of category membership for the missing categorical values.
<code>mincut</code>	If <code>imputed=TRUE</code> , there are categorical variables, and <code>impcat="rpart"</code> , <code>mincut</code> specifies the lowest node size that will be allowed to be split. The default is 40.
<code>inverse</code>	By default, imputed values are back-solved on the original scale using inverse linear interpolation on the fitted tabulated transformed values. This will cause distorted distributions of imputed values (e.g., floor and ceiling effects) when the estimated transformation has a flat or nearly flat section. To instead use the <code>invertTabulated</code> function (see above) with the <code>"sample"</code> option, specify <code>inverse="sample"</code> .
<code>tolInverse</code>	the multiplier of the range of transformed values, weighted by <code>freq</code> and by the distance measure, for determining the set of <code>x</code> values having <code>y</code> values within a tolerance of the value of <code>aty</code> in <code>invertTabulated</code> . For <code>predict.transcan</code> , <code>inverse</code> and <code>tolInverse</code> are obtained from options that were specified to <code>transcan</code> by default. Otherwise, if not specified by the user, these default to the defaults used to <code>invertTabulated</code> .

<code>pr</code>	For <code>transcan</code> , set to <code>FALSE</code> to suppress printing $R^2$ and shrinkage factors. Set <code>impute.transcan=FALSE</code> to suppress messages concerning the number of NA values imputed. Set <code>fit.mult.impute=FALSE</code> to suppress printing variance inflation factors accounting for imputation, rate of missing information, and degrees of freedom.
<code>pl</code>	Set to <code>FALSE</code> to suppress plotting the final transformations with distribution of scores for imputed values (if <code>show.na=TRUE</code> ).
<code>allpl</code>	Set to <code>TRUE</code> to plot transformations for intermediate iterations.
<code>show.na</code>	Set to <code>FALSE</code> to suppress the distribution of scores assigned to missing values (as tick marks on the right margin of each graph). See also <code>imputed</code> .
<code>imputed.actual</code>	The default is <code>"none"</code> to suppress plotting of actual vs. imputed values for all variables having any NA values. Other choices are <code>"datadensity"</code> to use <a href="#">datadensity</a> to make a single plot, <code>"hist"</code> to make a series of back-to-back histograms, <code>"qq"</code> to make a series of q-q plots, or <code>"ecdf"</code> to make a series of empirical cdfs. For <code>imputed.actual="datadensity"</code> for example you get a rug plot of the non-missing values for the variable with beneath it a rug plot of the imputed values. When <code>imputed.actual</code> is not <code>"none"</code> , <code>imputed</code> is automatically set to <code>TRUE</code> .
<code>iter.max</code>	maximum number of iterations to perform for <code>transcan</code> or <code>predict</code> . For <code>predict</code> , only one iteration is used if there are no NA values in the data or if <code>imp.con</code> was used.
<code>eps</code>	convergence criterion for <code>transcan</code> and <code>predict</code> . <code>eps</code> is the maximum change in transformed values from one iteration to the next. If for a given iteration all new transformations of variables differ by less than <code>eps</code> (with or without negating the transformation to allow for "flipping") from the transformations in the previous iteration, one more iteration is done for <code>transcan</code> . During this last iteration, individual transformations are not updated but coefficients of transformations are. This improves stability of coefficients of canonical variates on the right-hand-side. <code>eps</code> is ignored when <code>rhsImp="random"</code> .
<code>curtail</code>	for <code>transcan</code> , causes imputed values on the transformed scale to be truncated so that their ranges are within the ranges of non-imputed transformed values. For <code>predict</code> , <code>curtail</code> defaults to <code>TRUE</code> to truncate predicted transformed values to their ranges in the original fit ( <code>xt</code> ).
<code>imp.con</code>	for <code>transcan</code> , set to <code>TRUE</code> to impute NA values on the original scales with constants (medians or most frequent category codes). Set to a vector of constants to instead always use these constants for imputation. These imputed values are ignored when fitting the current working transformation for a single variable.
<code>shrink</code>	default is <code>FALSE</code> to use ordinary least squares or canonical variate estimates. For the purposes of imputing NAs, you may want to set <code>shrink=TRUE</code> to avoid overfitting when developing a prediction equation to predict each variables from all the others (see details below).
<code>init.cat</code>	method for initializing scorings of categorical variables. Default is <code>"mode"</code> to use a dummy variable set to 1 if the value is the most frequent value (this is the default). Use <code>"random"</code> to use a random 0-1 variable. Set to <code>"asis"</code> to use the original integer codes as starting scores.

nres	number of residuals to store if <code>n.impute</code> is specified. If the dataset has fewer than <code>nres</code> observations, all residuals are saved. Otherwise a random sample of the residuals of length <code>nres</code> without replacement is saved. The default for <code>nres</code> is higher if <code>boot.method="approximate bayesian"</code> .
data	Data frame used to fill the formula. For <code>ggplot</code> is the result of <code>transcan</code> with <code>trantab=TRUE</code> .
subset	an integer or logical vector specifying the subset of observations to fit
na.action	These may be used if <code>x</code> is a formula. The default <code>na.action</code> is <code>na.retain</code> (defined by <code>transcan</code> ) which keeps all observations with any NA values. For <code>impute.transcan</code> , <code>data</code> is a data frame to use as the source of variables to be imputed, rather than using <code>pos.in</code> . For <code>fit.mult.impute</code> , <code>data</code> is mandatory and is a data frame containing the data to be used in fitting the model but before imputations are applied. Variables omitted from <code>data</code> are assumed to be available from <code>frame1</code> and do not need to be imputed.
treeinfo	Set to <code>TRUE</code> to get additional information printed when <code>impcat="rpart"</code> , such as the predicted probabilities of category membership.
rhsImp	Set to <code>"random"</code> to use random draw imputation when a sometimes missing variable is moved to be a predictor of other sometimes missing variables. Default is <code>rhsImp="mean"</code> , which uses conditional mean imputation on the transformed scale. Residuals used are residuals from the transformed scale. When <code>"random"</code> is used, <code>transcan</code> runs 5 iterations and ignores eps.
details.impcat	set to a character scalar that is the name of a category variable to include in the resulting <code>transcan</code> object an element <code>details.impcat</code> containing details of how the categorical variable was multiply imputed.
...	arguments passed to <code>scat1d</code> . For <code>ggplot.transcan</code> , these arguments are passed to <code>facet_wrap</code> , e.g. <code>ncol=2</code> .
long	for <code>summary</code> , set to <code>TRUE</code> to print all imputed values. For <code>print</code> , set to <code>TRUE</code> to print details of transformations/imputations.
digits	number of significant digits for printing values by <code>summary</code>
scale	for <code>ggplot.transcan</code> set <code>scale=TRUE</code> to scale transformed values to <code>[0,1]</code> before plotting.
mapping, environment	not used; needed because of rules about generics
var	For <code>impute</code> , is a variable that was originally a column in <code>x</code> , for which imputed values are to be filled in. <code>imputed=TRUE</code> must have been used in <code>transcan</code> . Omit <code>var</code> to impute all variables, creating new variables in position <code>pos</code> (see <code>assign</code> ).
imputation	specifies which of the multiple imputations to use for filling in NA values
name	name of variable to impute, for <code>impute</code> function. Default is character string version of the second argument ( <code>var</code> ) in the call to <code>impute</code> . For <code>invertTabulated</code> , is the name of variable being transformed (used only for warning messages).
pos.in	location as defined by <code>assign</code> to find variables that need to be imputed, when all variables are to be imputed automatically by <code>impute.transcan</code> (i.e., when no input variable name is specified). Default is position that contains the first variable to be imputed.

<code>list.out</code>	If <code>var</code> is not specified, you can set <code>list.out=TRUE</code> to have <code>impute.transcan</code> return a list containing variables with needed values imputed. This list will contain a single imputation. Variables not needing imputation are copied to the list as-is. You can use this list for analysis just like a data frame.
<code>check</code>	set to <code>FALSE</code> to suppress certain warning messages
<code>newdata</code>	a new data matrix for which to compute transformed variables. Categorical variables must use the same integer codes as were used in the call to <code>transcan</code> . If a formula was originally specified to <code>transcan</code> (instead of a data matrix), <code>newdata</code> is optional and if given must be a data frame; a model frame is generated automatically from the previous formula. The <code>na.action</code> is handled automatically, and the levels for factor variables must be the same and in the same order as were used in the original variables specified in the formula given to <code>transcan</code> .
<code>fit.reps</code>	set to <code>TRUE</code> to save all fit objects from the fit for each imputation in <code>fit.mult.impute</code> . Then the object returned will have a component <code>fits</code> which is a list whose <i>i</i> 'th element is the <i>i</i> 'th fit object.
<code>dtrans</code>	provides an approach to creating derived variables from a single filled-in dataset. The function specified as <code>dtrans</code> can even reshape the imputed dataset. An example of such usage is fitting time-dependent covariates in a Cox model that are created by "start,stop" intervals. Imputations may be done on a one record per subject data frame that is converted by <code>dtrans</code> to multiple records per subject. The imputation can enforce consistency of certain variables across records so that for example a missing value of sex will not be imputed as 'male' for one of the subject's records and 'female' as another. An example of how <code>dtrans</code> might be specified is <code>dtrans=function(w) {w\$age &lt;- w\$years + w\$months/12; w}</code> where <code>months</code> might have been imputed but <code>years</code> was never missing. An outline for using 'dtrans' to impute missing baseline variables in a longitudinal analysis appears in Details below.
<code>derived</code>	an expression containing R expressions for computing derived variables that are used in the model formula. This is useful when multiple imputations are done for component variables but the actual model uses combinations of these (e.g., ratios or other derivations). For a single derived variable you can specify for example <code>derived=expression(ratio &lt;- weight/height)</code> . For multiple derived variables use the form <code>derived=expression({ratio &lt;- weight/height; product &lt;- weight*height})</code> or put the expression on separate input lines. To monitor the multiply-imputed derived variables you can add to the expression a command such as <code>print(describe(ratio))</code> . See the example below. Note that <code>derived</code> is not yet implemented.
<code>fun</code>	a function of a fit made on one of the completed datasets. Typical uses are bootstrap model validations. The result of <code>fun</code> for imputation <i>i</i> is placed in the <i>i</i> th element of a list that is returned in the <code>fit.mult.impute</code> object element named <code>funresults</code> . See the <code>rms</code> <code>processMI</code> function for help in processing these results for the cases of <code>validate</code> and <code>calibrate</code> .
<code>vcovOpts</code>	a list of named additional arguments to pass to the <code>vcov</code> method for <code>fitter</code> . Useful for <code>orm</code> models for retaining all intercepts ( <code>vcovOpts=list(intercepts='all')</code> ) instead of just the middle one.

robust	set to TRUE to have <code>fit.mult.impute</code> call the rms package <code>robcov</code> function on each fit on a completed dataset. When <code>cluster</code> is given, <code>robust</code> is forced to TRUE.
cluster	a vector of cluster IDs that is the same length of the number of rows in the dataset being analyzed. When specified, <code>robust</code> is assumed to be TRUE, and the rms <code>robcov</code> function is called with the <code>cluster</code> vector given as its second argument.
robmethod	see the <code>robcov</code> function's <code>method</code> argument
funstack	set to FALSE to not run <code>fun</code> on the stacked dataset, making an <code>n.impute+1</code> element of <code>funresults</code>
lrt	set to TRUE to have <code>method</code> , <code>fun</code> , <code>fitargs</code> set appropriately automatically so that <code>processMI</code> can be used to get likelihood ratio tests. When doing this, <code>fun</code> may not be specified by the user.
fitargs	a list of extra arguments to pass to <code>fitter</code> , used especially with <code>fun</code> . When <code>robust=TRUE</code> the arguments <code>x=TRUE</code> , <code>y=TRUE</code> are automatically added to <code>fitargs</code> .
type	By default, the matrix of transformed variables is returned, with imputed values on the transformed scale. If you had specified <code>trantab=TRUE</code> to <code>transcan</code> , specifying <code>type="original"</code> does the table look-ups with linear interpolation to return the input matrix <code>x</code> but with imputed values on the original scale inserted for NA values. For categorical variables, the method used here is to select the category code having a corresponding scaled value closest to the predicted transformed value. This corresponds to the default <code>impcat</code> . Note: imputed values thus returned when <code>type="original"</code> are single expected value imputations even in <code>n.impute</code> is given.
object	an object created by <code>transcan</code> , or an object to be converted to R function code, typically a model fit object of some sort
prefix, suffix	When creating separate R functions for each variable in <code>x</code> , the name of the new function will be <code>prefix</code> placed in front of the variable name, and <code>suffix</code> placed in back of the name. The default is to use names of the form <code>' .varname'</code> , where <code>varname</code> is the variable name.
pos	position as in <code>assign</code> at which to store new functions (for <code>Function</code> ). Default is <code>pos=-1</code> .
y	a vector corresponding to <code>x</code> for <code>invertTabulated</code> , if its first argument <code>x</code> is not a list
freq	a vector of frequencies corresponding to cross-classified <code>x</code> and <code>y</code> if <code>x</code> is not a list. Default is a vector of ones.
aty	vector of transformed values at which inverses are desired
rule	see <code>approx</code> . <code>transcan</code> assumes <code>rule</code> is always 2.
regcoef.only	set to TRUE to make <code>vcov.default</code> delete positions in the covariance matrix for any non-regression coefficients (e.g., log scale parameter from <code>psm</code> or <code>survreg</code> )
intercepts	this is primarily for <code>orm</code> objects. Set to <code>"none"</code> to discard all intercepts from the covariance matrix, or to <code>"all"</code> or <code>"mid"</code> to keep all elements generated by <code>orm</code> ( <code>orm</code> only outputs the covariance matrix for the intercept corresponding to the median). You can also set <code>intercepts</code> to a vector of subscripts for selecting particular intercepts in a multi-intercept model.

## Details

The starting approximation to the transformation for each variable is taken to be the original coding of the variable. The initial approximation for each missing value is taken to be the median of the non-missing values for the variable (for continuous ones) or the most frequent category (for categorical ones). Instead, if `imp.con` is a vector, its values are used for imputing NA values. When using each variable as a dependent variable, NA values on that variable cause all observations to be temporarily deleted. Once a new working transformation is found for the variable, along with a model to predict that transformation from all the other variables, that latter model is used to impute NA values in the selected dependent variable if `imp.con` is not specified.

When that variable is used to predict a new dependent variable, the current working imputed values are inserted. Transformations are updated after each variable becomes a dependent variable, so the order of variables on `x` could conceivably make a difference in the final estimates. For obtaining out-of-sample predictions/transformations, `predict` uses the same iterative procedure as `transcan` for imputation, with the same starting values for fill-ins as were used by `transcan`. It also (by default) uses a conservative approach of curtailing transformed variables to be within the range of the original ones. Even when `method = "pc"` is specified, canonical variables are used for imputing missing values.

Note that fitted transformations, when evaluated at imputed variable values (on the original scale), will not precisely match the transformed imputed values returned in `xt`. This is because `transcan` uses an approximate method based on linear interpolation to back-solve for imputed values on the original scale.

`Shrinkage` uses the method of *Van Houwelingen and Le Cessie (1990)* (similar to *Copas, 1983*). The shrinkage factor is

$$1 - \frac{(1-R^2)(n-1)}{n-k-1} \frac{1}{R^2}$$

where  $R^2$  is the apparent  $R^2$  for predicting the variable,  $n$  is the number of non-missing values, and  $k$  is the effective number of degrees of freedom (aside from intercepts). A heuristic estimate is used for  $k$ :  $A - 1 + \text{sum}(\max(0, B_i - 1)) / m + m$ , where  $A$  is the number of d.f. required to represent the variable being predicted, the  $B_i$  are the number of columns required to represent all the other variables, and  $m$  is the number of all other variables. Division by  $m$  is done because the transformations for the other variables are fixed at their current transformations the last time they were being predicted. The  $+m$  term comes from the number of coefficients estimated on the right hand side, whether by least squares or canonical variates. If a shrinkage factor is negative, it is set to 0. The shrinkage factor is the ratio of the adjusted  $R^2$  to the ordinary  $R^2$ . The adjusted  $R^2$  is

$$1 - \frac{(1 - R^2)(n - 1)}{n - k - 1}$$

which is also set to zero if it is negative. If `shrink=FALSE` and the adjusted  $R^2$ s are much smaller than the ordinary  $R^2$ s, you may want to run `transcan` with `shrink=TRUE`.

Canonical variates are scaled to have variance of 1.0, by multiplying canonical coefficients from `cancor` by  $\sqrt{n - 1}$ .

When specifying a non-**rms** library fitting function to `fit.mult.impute` (e.g., `lm`, `glm`), running the result of `fit.mult.impute` through that fit's `summary` method will not use the imputation-adjusted variances. You may obtain the new variances using `fit$var` or `vcov(fit)`.

When you specify a **rms** function to `fit.mult.impute` (e.g. `lrm`, `ols`, `cph`, `psm`, `bj`, `Rq`, `Gls`, `Glm`), automatically computed transformation parameters (e.g., knot locations for `rcs`) that are estimated

for the first imputation are used for all other imputations. This ensures that knot locations will not vary, which would change the meaning of the regression coefficients.

Warning: even though `fit.mult.impute` takes imputation into account when estimating variances of regression coefficient, it does not take into account the variation that results from estimation of the shapes and regression coefficients of the customized imputation equations. Specifying `shrink=TRUE` solves a small part of this problem. To fully account for all sources of variation you should consider putting the `transcan` invocation inside a bootstrap or loop, if execution time allows. Better still, use `aregImpute` or a package such as `mice` that uses real Bayesian posterior realizations to multiply impute missing values correctly.

It is strongly recommended that you use the `Hmisc naclus` function to determine if there is a good basis for imputation. `naclus` will tell you, for example, if systolic blood pressure is missing whenever diastolic blood pressure is missing. If the only variable that is well correlated with diastolic bp is systolic bp, there is no basis for imputing diastolic bp in this case.

At present, `predict` does not work with multiple imputation.

When calling `fit.mult.impute` with `glm` as the fitter argument, if you need to pass a family argument to `glm` do it by quoting the family, e.g., `family="binomial"`.

`fit.mult.impute` will not work with proportional odds models when regression imputation was used (as opposed to predictive mean matching). That's because regression imputation will create values of the response variable that did not exist in the dataset, altering the intercept terms in the model.

You should be able to use a variable in the formula given to `fit.mult.impute` as a numeric variable in the regression model even though it was a factor variable in the invocation of `transcan`. Use for example `fit.mult.impute(y ~ codes(x), lrm, trans)` (thanks to Trevor Thompson <trevor@hp5.eushc.org>).

Here is an outline of the steps necessary to impute baseline variables using the `dtrans` argument, when the analysis to be repeated by `fit.mult.impute` is a longitudinal analysis (using e.g. GLs).

1. Create a one row per subject data frame containing baseline variables plus follow-up variables that are assigned to windows. For example, you may have dozens of repeated measurements over years but you capture the measurements at the times measured closest to 1, 2, and 3 years after study entry
2. Make sure the dataset contains the subject ID
3. This dataset becomes the one passed to `aregImpute` as `data=`. You will be imputing missing baseline variables from follow-up measurements defined at fixed times.
4. Have another dataset with all the non-missing follow-up values on it, one record per measurement time per subject. This dataset should not have the baseline variables on it, and the follow-up measurements should not be named the same as the baseline variable(s); the subject ID must also appear
5. Add the `dtrans` argument to `fit.mult.impute` to define a function with one argument representing the one record per subject dataset with missing values filled it from the current imputation. This function merges the above 2 datasets; the returned value of this function is the merged data frame.
6. This merged-on-the-fly dataset is the one handed by `fit.mult.impute` to your fitting function, so variable names in the formula given to `fit.mult.impute` must match the names created by the merge

**Value**

For transcan, a list of class 'transcan' with elements

call	(with the function call)
iter	(number of iterations done)
rsq, rsq.adj	containing the $R^2$ s and adjusted $R^2$ s achieved in predicting each variable from all the others
categorical	the values supplied for categorical
asis	the values supplied for asis
coef	the within-variable coefficients used to compute the first canonical variate
xcoef	the (possibly shrunk) across-variables coefficients of the first canonical variate that predicts each variable in-turn.
parms	the parameters of the transformation (knots for splines, contrast matrix for categorical variables)
fillin	the initial estimates for missing values (NA if variable never missing)
ranges	the matrix of ranges of the transformed variables (min and max in first and second row)
scale	a vector of scales used to determine convergence for a transformation.
formula	the formula (if x was a formula)

, and optionally a vector of shrinkage factors used for predicting each variable from the others. For asis variables, the scale is the average absolute difference about the median. For other variables it is unity, since canonical variables are standardized. For xcoef, row i has the coefficients to predict transformed variable i, with the column for the coefficient of variable i set to NA. If imputed=TRUE was given, an optional element imputed also appears. This is a list with the vector of imputed values (on the original scale) for each variable containing NAs. Matrices rather than vectors are returned if n.impute is given. If trantab=TRUE, the trantab element also appears, as described above. If n.impute > 0, transcan also returns a list residuals that can be used for future multiple imputation.

impute returns a vector (the same length as var) of class 'impute' with NA values imputed.

predict returns a matrix with the same number of columns or variables as were in x.

fit.mult.impute returns a fit object that is a modification of the fit object created by fitting the completed dataset for the final imputation. The var matrix in the fit object has the imputation-corrected variance-covariance matrix. coefficients is the average (over imputations) of the coefficient vectors, variance.inflation.impute is a vector containing the ratios of the diagonals of the between-imputation variance matrix to the diagonals of the average apparent (within-imputation) variance matrix. missingInfo is *Rubin's rate of missing information* and dfmi is *Rubin's degrees of freedom for a t-statistic* for testing a single parameter. The last two objects are vectors corresponding to the diagonal of the variance matrix. The class "fit.mult.impute" is prepended to the other classes produced by the fitting function.

When method is not 'ordinary', i.e., stacking is used, fit.mult.impute returns a modified fit object that is computed on all completed datasets combined, with most all statistics that are functions of the sample size corrected to the real sample size. Elements in the fit such as residuals will have length equal to the real sample size times the number of imputations.

fit.mult.impute stores intercepts attributes in the coefficient matrix and in var for orm fits.



**Side Effects**

prints, plots, and impute. transcan creates new variables.

**Author(s)**

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>

**References**

- Kuhfeld, Warren F: The PRINQUAL Procedure. SAS/STAT User's Guide, Fourth Edition, Volume 2, pp. 1265–1323, 1990.
- Van Houwelingen JC, Le Cessie S: Predictive value of statistical models. *Statistics in Medicine* 8:1303–1325, 1990.
- Copas JB: Regression, prediction and shrinkage. *JRSS B* 45:311–354, 1983.
- He X, Shen L: Linear regression after spline transformation. *Biometrika* 84:474–481, 1997.
- Little RJA, Rubin DB: *Statistical Analysis with Missing Data*. New York: Wiley, 1987.
- Rubin DJ, Schenker N: Multiple imputation in health-care databases: An overview and some applications. *Stat in Med* 10:585–598, 1991.
- Faris PD, Ghali WA, et al: Multiple imputation versus data enhancement for dealing with missing data in observational health care outcome analyses. *J Clin Epidem* 55:184–191, 2002.

**See Also**

[aregImpute](#), [impute](#), [naclus](#), [naplot](#), [ace](#), [avas](#), [cancor](#), [prcomp](#), [rcspline.eval](#), [lsfit](#), [approx](#), [datadensity](#), [mice](#), [ggplot](#), [processMI](#)

**Examples**

```
## Not run:
x <- cbind(age, disease, blood.pressure, pH)
#cbind will convert factor object `disease' to integer
par(mfrow=c(2,2))
x.trans <- transcan(x, categorical="disease", asis="pH",
                  transformed=TRUE, imputed=TRUE)
summary(x.trans) #Summary distribution of imputed values, and R-squares
f <- lm(y ~ x.trans$transformed) #use transformed values in a regression
#Now replace NAs in original variables with imputed values, if not
#using transformations
age <- impute(x.trans, age)
disease <- impute(x.trans, disease)
blood.pressure <- impute(x.trans, blood.pressure)
pH <- impute(x.trans, pH)
#Do impute(x.trans) to impute all variables, storing new variables under
#the old names
summary(pH) #uses summary.impute to tell about imputations
```

```

#and summary.default to tell about pH overall
# Get transformed and imputed values on some new data frame xnew
newx.trans  <- predict(x.trans, xnew)
w           <- predict(x.trans, xnew, type="original")
age        <- w[, "age"]           #inserts imputed values
blood.pressure <- w[, "blood.pressure"]
Function(x.trans) #creates .age, .disease, .blood.pressure, .pH()
#Repeat first fit using a formula
x.trans <- transcan(~ age + disease + blood.pressure + I(pH),
                   imputed=TRUE)
age <- impute(x.trans, age)
predict(x.trans, expand.grid(age=50, disease="pneumonia",
                             blood.pressure=60:260, pH=7.4))
z <- transcan(~ age + factor(disease.code), # disease.code categorical
              transformed=TRUE, trantab=TRUE, imputed=TRUE, pl=FALSE)
ggplot(z, scale=TRUE)
plot(z$transformed)

## End(Not run)

# Multiple imputation and estimation of variances and covariances of
# regression coefficient estimates accounting for imputation
set.seed(1)
x1 <- factor(sample(c('a', 'b', 'c'), 100, TRUE))
x2 <- (x1=='b') + 3*(x1=='c') + rnorm(100)
y  <- x2 + 1*(x1=='c') + rnorm(100)
x1[1:20] <- NA
x2[18:23] <- NA
d <- data.frame(x1, x2, y)
n <- naclus(d)
plot(n); naplot(n) # Show patterns of NAs
f <- transcan(~y + x1 + x2, n.impute=10, shrink=FALSE, data=d)
options(digits=3)
summary(f)

f <- transcan(~y + x1 + x2, n.impute=10, shrink=TRUE, data=d)
summary(f)

h <- fit.mult.impute(y ~ x1 + x2, lm, f, data=d)
# Add ,fit.reps=TRUE to save all fit objects in h, then do something like:
# for(i in 1:length(h$fits)) print(summary(h$fits[[i]]))

diag(vcov(h))

h.complete <- lm(y ~ x1 + x2, na.action=na.omit)
h.complete
diag(vcov(h.complete))

```

```

# Note: had the rms ols function been used in place of lm, any
# function run on h (anova, summary, etc.) would have automatically
# used imputation-corrected variances and covariances

# Example demonstrating how using the multinomial logistic model
# to impute a categorical variable results in a frequency
# distribution of imputed values that matches the distribution
# of non-missing values of the categorical variable

## Not run:
set.seed(11)
x1 <- factor(sample(letters[1:4], 1000,TRUE))
x1[1:200] <- NA
table(x1)/sum(table(x1))
x2 <- runif(1000)
z <- transcan(~ x1 + I(x2), n.impute=20, impcat='multinom')
table(z$imputed$x1)/sum(table(z$imputed$x1))

# Here is how to create a completed dataset
d <- data.frame(x1, x2)
z <- transcan(~x1 + I(x2), n.impute=5, data=d)
imputed <- impute(z, imputation=1, data=d,
                 list.out=TRUE, pr=FALSE, check=FALSE)
sapply(imputed, function(x)sum(is.imputed(x)))
sapply(imputed, function(x)sum(is.na(x)))

## End(Not run)

# Do single imputation and create a filled-in data frame
z <- transcan(~x1 + I(x2), data=d, imputed=TRUE)
imputed <- as.data.frame(impute(z, data=d, list.out=TRUE))

# Example where multiple imputations are for basic variables and
# modeling is done on variables derived from these

set.seed(137)
n <- 400
x1 <- runif(n)
x2 <- runif(n)
y <- x1*x2 + x1/(1+x2) + rnorm(n)/3
x1[1:5] <- NA
d <- data.frame(x1,x2,y)
w <- transcan(~ x1 + x2 + y, n.impute=5, data=d)
# Add ,show.imputed.actual for graphical diagnostics
## Not run:
g <- fit.mult.impute(y ~ product + ratio, ols, w,
                    data=data.frame(x1,x2,y),
                    derived=expression({
                        product <- x1*x2

```

```

ratio <- x1/(1+x2)
print(cbind(x1,x2,x1*x2,product)[1:6,]))

## End(Not run)

# Here's a method for creating a permanent data frame containing
# one set of imputed values for each variable specified to transcan
# that had at least one NA, and also containing all the variables
# in an original data frame. The following is based on the fact
# that the default output location for impute.transcan is
# given by the global environment

## Not run:
xt <- transcan(~. , data=mine,
               imputed=TRUE, shrink=TRUE, n.impute=10, trantab=TRUE)
attach(mine, use.names=FALSE)
impute(xt, imputation=1) # use first imputation
# omit imputation= if using single imputation
detach(1, 'mine2')

## End(Not run)

# Example of using invertTabulated outside transcan
x <- c(1,2,3,4,5,6,7,8,9,10)
y <- c(1,2,3,4,5,5,5,5,9,10)
freq <- c(1,1,1,1,1,2,3,4,1,1)
# x=5,6,7,8 with prob. .1 .2 .3 .4 when y=5
# Within a tolerance of .05*(10-1) all y's match exactly
# so the distance measure does not play a role
set.seed(1) # so can reproduce
for(inverse in c('linearInterp','sample')) {
  print(table(invertTabulated(x, y, freq, rep(5,1000), inverse=inverse)))

# Test inverse='sample' when the estimated transformation is
# flat on the right. First show default imputations
set.seed(3)
x <- rnorm(1000)
y <- pmin(x, 0)
x[1:500] <- NA
for(inverse in c('linearInterp','sample')) {
  par(mfrow=c(2,2))
  w <- transcan(~ x + y, imputed.actual='hist',
                inverse=inverse, curtail=FALSE,
                data=data.frame(x,y))
  if(inverse=='sample') next
# cat('Click mouse on graph to proceed\n')
# locator(1)
}

```

```

## Not run:
# While running multiple imputation for a logistic regression model
# Run the rms package validate and calibrate functions and save the
# results in w$funresults
a <- aregImpute(~ x1 + x2 + y, data=d, n.impute=10)
require(rms)
g <- function(fit)
  list(validate=validate(fit, B=50), calibrate=calibrate(fit, B=75))
w <- fit.mult.impute(y ~ x1 + x2, lrm, a, data=d, fun=g,
  fitargs=list(x=TRUE, y=TRUE))
# Get all validate results in it's own list of length 10
r <- w$funresults
val <- lapply(r, function(x) x$validate)
cal <- lapply(r, function(x) x$calibrate)
# See rms processMI and https://hbiostat.org/rmsc/validate.html#sec-val-mival

## End(Not run)

## Not run:
# Account for within-subject correlation using the robust cluster sandwich
# covariance estimate in conjunction with Rubin's rule for multiple imputation
# rms package must be installed
a <- aregImpute(..., data=d)
f <- fit.mult.impute(y ~ x1 + x2, lrm, a, n.impute=30, data=d, cluster=d$id)
# Get likelihood ratio chi-square tests accounting for missingness
a <- aregImpute(..., data=d)
h <- fit.mult.impute(y ~ x1 + x2, lrm, a, n.impute=40, data=d, lrt=TRUE)
processMI(h, which='anova') # processMI is in rms

## End(Not run)

```

---

translate

*Translate Vector or Matrix of Text Strings*


---

## Description

Uses the UNIX `tr` command to translate any character in `old` in `text` to the corresponding character in `new`. If `multichar=T` or `old` and `new` have more than one element, or each have one element but they have different numbers of characters, uses the UNIX `sed` command to translate the series of characters in `old` to the series in `new` when these characters occur in `text`. If `old` or `new` contain a backslash, you sometimes have to quadruple it to make the UNIX command work. If they contain a forward slash, precede it by two backslashes. Invokes the builtin `chartr` function if `multichar=FALSE`.

## Usage

```
translate(text, old, new, multichar=FALSE)
```

**Arguments**

text            scalar, vector, or matrix of character strings to translate.  
 old            vector old characters  
 new            corresponding vector of new characters  
 multichar      See above.

**Value**

an object like text but with characters translated

**See Also**

grep

**Examples**

```
translate(c("ABC","DEF"),"ABCDEFG", "abcdefg")
translate("23.12","[.]", "\\cdot ") # change . to \cdot
translate(c("dog","cat","tiger"),c("dog","cat"),c("DOG","CAT"))
# S-Plus gives [1] "DOG" "CAT" "tiger" - check discrepancy
translate(c("dog","cat2","snake"),c("dog","cat"),"animal")
# S-Plus gives [1] "animal" "animal2" "snake"
```

---

trunc.POSIXt	<i>Return the floor, ceiling, or rounded value of date or time to specified unit.</i>
--------------	---

---

**Description**

truncPOSIXt returns the date truncated to the specified unit. ceil.POSIXt returns next ceiling of the date at the unit selected in units. roundPOSIXt returns the date or time value rounded to nearest specified unit selected in digits.

truncPOSIXt and roundPOSIXt have been extended from the base package functions trunc.POSIXt and round.POSIXt which in the future will add the other time units we need.

**Usage**

```
ceil(x, units,...)
## Default S3 method:
ceil(x, units, ...)
truncPOSIXt(x, units = c("secs", "mins", "hours", "days",
"months", "years"), ...)
## S3 method for class 'POSIXt'
ceil(x, units = c("secs", "mins", "hours", "days",
"months", "years"), ...)
roundPOSIXt(x, digits = c("secs", "mins", "hours", "days", "months", "years"))
```

**Arguments**

`x` date to be ceilinged, truncated, or rounded  
`units` unit to that is rounded up or down to.  
`digits` same as `units` but different name to be compatible with `round` generic.  
`...` further arguments to be passed to or from other methods.

**Value**

An object of class `POSIXlt`.

**Author(s)**

Charles Dupont

**See Also**

[Date POSIXt POSIXlt DateTimeClasses](#)

**Examples**

```
date <- ISOdate(1832, 7, 12)
ceil(date, units='months') # '1832-8-1'
truncPOSIXt(date, units='years') # '1832-1-1'
roundPOSIXt(date, digits='months') # '1832-7-1'
```

---

units

*Units Attribute of a Vector*

---

**Description**

Sets or retrieves the "units" attribute of an object. For `units.default` replaces the builtin version, which only works for time series objects. If the variable is also given a label, subsetting (using `[.labelled]`) will retain the "units" attribute. For a `Surv` object, `units` first looks for an overall "units" attribute, then it looks for `units` for the `time2` variable then for `time1`.

**Usage**

```
units(x, ...)
## Default S3 method:
units(x, none='', ...)
## S3 method for class 'Surv'
units(x, none='', ...)
## Default S3 replacement method:
units(x) <- value
```

**Arguments**

x	any object
...	ignored
value	the units of the object, or ""
none	value to which to set result if no appropriate attribute is found

**Value**

the units attribute of x, if any; otherwise, the units attribute of the `tspar` attribute of x if any; otherwise the value `none`. Handling for `Surv` objects is different (see above).

**See Also**

[label](#)

**Examples**

```
require(survival)
fail.time <- c(10,20)
units(fail.time) <- "Day"
describe(fail.time)
S <- Surv(fail.time)
units(S)

label(fail.time) <- 'Failure Time'
fail.time
```

---

upData

---

*Update a Data Frame or Cleanup a Data Frame after Importing*


---

**Description**

`cleanup.import` will correct errors and shrink the size of data frames. By default, double precision numeric variables are changed to integer when they contain no fractional components. Infinite values or values greater than  $1e20$  in absolute value are set to NA. This solves problems of importing Excel spreadsheets that contain occasional character values for numeric columns, as S converts these to Inf without warning. There is also an option to convert variable names to lower case and to add labels to variables. The latter can be made easier by importing a CNTLOUT dataset created by SAS PROC FORMAT and using the `sasdict` option as shown in the example below. `cleanup.import` can also transform character or factor variables to dates.

`upData` is a function facilitating the updating of a data frame without attaching it in search position one. New variables can be added, old variables can be modified, variables can be removed or renamed, and "labels" and "units" attributes can be provided. Observations can be subsetted. Various checks are made for errors and inconsistencies, with warnings issued to help the user. Levels of factor variables can be replaced, especially using the `list` notation of the standard `merge.levels` function. Unless `force.single` is set to FALSE, `upData` also converts double precision vectors to



integer if no fractional values are present in a vector. `upData` is also used to process R workspace objects created by `StatTransfer`, which puts variable and value labels as attributes on the data frame rather than on each variable. If such attributes are present, they are used to define all the labels and value labels (through conversion to factor variables) before any label changes take place, and `force.single` is set to a default of `FALSE`, as `StatTransfer` already does conversion to integer.

Variables having labels but not classed "labelled" (e.g., data imported using the haven package) have that class added to them by `upData`.

The `dataframeReduce` function removes variables from a data frame that are problematic for certain analyses. Variables can be removed because the fraction of missing values exceeds a threshold, because they are character or categorical variables having too many levels, or because they are binary and have too small a prevalence in one of the two values. Categorical variables can also have their levels combined when a level is of low prevalence. A data frame listing actions take is return as attribute "info" to the main returned data frame.

## Usage

```
cleanup.import(obj, labels, lowernames=FALSE,
               force.single=TRUE, force.numeric=TRUE, rmnames=TRUE,
               big=1e20, sasdict, print, datevars=NULL, datetimevars=NULL,
               dateformat='%F',
               fixdates=c('none', 'year'),
               autodate=FALSE, autonum=FALSE, fracnn=0.3,
               considerNA=NULL, charfactor=FALSE)

upData(object, ...,
        subset, rename, drop, keep, labels, units, levels, force.single=TRUE,
        lowernames=FALSE, caplabels=FALSE, moveUnits=FALSE,
        charfactor=FALSE, print=TRUE, html=FALSE)

dataframeReduce(data, fracmiss=1, maxlevels=NULL, minprev=0, print=TRUE)
```

## Arguments

<code>obj</code>	a data frame or list
<code>object</code>	a data frame or list
<code>data</code>	a data frame
<code>force.single</code>	By default, double precision variables are converted to single precision (in S-Plus only) unless <code>force.single=FALSE</code> . <code>force.single=TRUE</code> will also convert vectors having only integer values to have a storage mode of integer, in R or S-Plus.
<code>force.numeric</code>	Sometimes importing will cause a numeric variable to be changed to a factor vector. By default, <code>cleanup.import</code> will check each factor variable to see if the levels contain only numeric values and "". In that case, the variable will be converted to numeric, with "" converted to NA. Set <code>force.numeric=FALSE</code> to prevent this behavior.
<code>rmnames</code>	set to 'F' to not have 'cleanup.import' remove 'names' or '.Names' attributes from variables

labels	a character vector the same length as the number of variables in obj. These character values are taken to be variable labels in the same order of variables in obj. For upData, labels is a named list or named vector with variables in no specific order.
lowernames	set this to TRUE to change variable names to lower case. upData does this before applying any other changes, so variable names given inside arguments to upData need to be lower case if lowernames==TRUE.
big	a value such that values larger than this in absolute value are set to missing by cleanup.import
sasdict	the name of a data frame containing a raw imported SAS PROC CONTENTS CNTLOUT= dataset. This is used to define variable names and to add attributes to the new data frame specifying the original SAS dataset name and label.
print	set to TRUE or FALSE to force or prevent printing of the current variable number being processed. By default, such messages are printed if the product of the number of variables and number of observations in obj exceeds 500,000. For dataframeReduce set print to FALSE to suppress printing information about dropped or modified variables. Similar for upData.
datevars	character vector of names (after lowernames is applied) of variables to consider as a factor or character vector containing dates in a format matching dateformat. The default is "%F" which uses the yyyy-mm-dd format.
datetimevars	character vector of names (after lowernames is applied) of variables to consider to be date-time variables, with date formats as described under datevars followed by a space followed by time in hh:mm:ss format. chron is used to store date-time variables. If all times in the variable are 00:00:00 the variable will be converted to an ordinary date variable.
dateformat	for cleanup.import is the input format (see <a href="#">strptime</a> )
fixdates	for any of the variables listed in datevars that have a dateformat that cleanup.import understands, specifying fixdates allows corrections of certain formatting inconsistencies before the fields are attempted to be converted to dates (the default is to assume that the dateformat is followed for all observation for datevars). Currently fixdates='year' is implemented, which will cause 2-digit or 4-digit years to be shifted to the alternate number of digits when dateform is the default "%F" or is "%y-%m-%d", "%m/%d/%y", or "%m/%d/%Y". Two-digits years are padded with 20 on the left. Set dateformat to the desired format, not the exceptional format.
autodate	set to TRUE to have cleanup.import determine and automatically handle factor or character vectors that mainly contain dates of the form YYYY-mm-dd, mm/dd/YYYY, YYYY, or mm/YYYY, where the later two are imputed to, respectively, July 3 and the 15th of the month. Takes effect when the fraction of non-dates (of non-missing values) is less than fracnn to allow for some free text such as "unknown". Attributes special.miss and imputed are created for the vector so that describe() will inform the user. Illegal values are converted to NAs and stored in the special.miss attribute.
autonum	set to TRUE to have cleanup.import examine (after autodate) character and factor variables to see if they are legal numerics exact for at most a fraction of fracnn of non-missing non-numeric values. Qualifying variables are converted

	to numeric, and illegal values set to NA and stored in the special <code>.miss</code> attribute to enhance describe output.
<code>fracnn</code>	see <code>autodate</code> and <code>autonum</code>
<code>considerNA</code>	for <code>autodate</code> and <code>autonum</code> , considers character values in the vector <code>considerNA</code> to be the same as NA. Leading and trailing white space and upper/lower case are ignored.
<code>charfactor</code>	set to TRUE to change character variables to factors if they have fewer than $n/2$ unique values. Null strings and blanks are converted to NAs.
<code>...</code>	for <code>upData</code> , one or more expressions of the form <code>variable=expression</code> , to derive new variables or change old ones.
<code>subset</code>	an expression that evaluates to a logical vector specifying which rows of object should be retained. The expressions should use the original variable names, i.e., before any variables are renamed but after <code>lowernames</code> takes effect.
<code>rename</code>	list or named vector specifying old and new names for variables. Variables are renamed before any other operations are done. For example, to rename variables <code>age</code> and <code>sex</code> to respectively <code>Age</code> and <code>gender</code> , specify <code>rename=list(age="Age", sex="gender")</code> or <code>rename=c(age=...)</code> .
<code>drop</code>	a vector of variable names to remove from the data frame
<code>keep</code>	a vector of variable names to keep, with all other variables dropped
<code>units</code>	a named vector or list defining "units" attributes of variables, in no specific order
<code>levels</code>	a named list defining "levels" attributes for factor variables, in no specific order. The values in this list may be character vectors redefining levels (in order) or another list (see <code>merge.levels</code> if using S-Plus).
<code>caplabels</code>	set to TRUE to capitalize the first letter of each word in each variable label
<code>moveUnits</code>	set to TRUE to look for units of measurements in variable labels and move them to a "units" attribute. If an expression in a label is enclosed in parentheses or brackets it is assumed to be units if <code>moveUnits=TRUE</code> .
<code>html</code>	set to TRUE to print conversion information as <code>html</code> verbatim at 0.6 size. The user will need to put <code>results='asis'</code> in a knitr chunk header to properly render this output.
<code>fracmiss</code>	the maximum permissible proportion of NAs for a variable to be kept. Default is to keep all variables no matter how many NAs are present.
<code>maxlevels</code>	the maximum number of levels of a character or categorical or factor variable before the variable is dropped
<code>minprev</code>	the minimum proportion of non-missing observations in a category for a binary variable to be retained, and the minimum relative frequency of a category before it will be combined with other small categories

**Value**

a new data frame

**Author(s)**

Frank Harrell, Vanderbilt University

**See Also**

[sas.get](#), [data.frame](#), [describe](#), [label](#), [read.csv](#), [strptime](#), [POSIXct](#), [Date](#)

**Examples**

```
## Not run:
dat <- read.table('myfile.asc')
dat <- cleanup.import(dat)

## End(Not run)
dat <- data.frame(a=1:3, d=c('01/02/2004', '1/3/04', ''))
cleanup.import(dat, datevars='d', dateformat='%m/%d/%y', fixdates='year')

dat <- data.frame(a=(1:3)/7, y=c('a', 'b1', 'b2'), z=1:3)
dat2 <- upData(dat, x=x^2, x=x-5, m=x/10,
              rename=c(a='x'), drop='z',
              labels=c(x='X', y='test'),
              levels=list(y=list(a='a', b=c('b1', 'b2'))))

dat2
describe(dat2)
dat <- dat2 # copy to original name and delete dat2 if OK
rm(dat2)
dat3 <- upData(dat, X=X^2, subset = x < (3/7)^2 - 5, rename=c(x='X'))

# Remove hard to analyze variables from a redundancy analysis of all
# variables in the data frame
d <- dataframeReduce(dat, fracmiss=.1, minprev=.05, maxlevels=5)
# Could run redun(~., data=d) at this point or include dataframeReduce
# arguments in the call to redun

# If you import a SAS dataset created by PROC CONTENTS CNTLOUT=x.datadict,
# the LABELS from this dataset can be added to the data. Let's also
# convert names to lower case for the main data file
## Not run:
mydata2 <- cleanup.import(mydata2, lowernames=TRUE, sasdict=datadict)

## End(Not run)
```

---

upFirst

*Change First Letters to Upper Case*

---

**Description**

Changes the first letter of each word in a string to upper case, keeping selected words in lower case. Words containing at least 2 capital letters are kept as-is.

**Usage**

```
upFirst(txt, lower = FALSE, alllower = FALSE)
```

**Arguments**

txt	a character vector
lower	set to TRUE to make only the very first letter of the string upper case, and to keep words with at least 2 capital letters in their original form
alllower	set to TRUE to make every word start with lower case unless it has at least 2 caps

**References**

[https://en.wikipedia.org/wiki/Letter\\_case#Headings\\_and\\_publication\\_titles](https://en.wikipedia.org/wiki/Letter_case#Headings_and_publication_titles)

**Examples**

```
upFirst(c('this and that', 'that is Beyond question'))
```

---

valueTags	<i>Store Descriptive Information About an Object</i>
-----------	--

---

**Description**

Functions get or set useful information about the contents of the object for later use.

**Usage**

```
valueTags(x)  
valueTags(x) <- value  
  
valueLabel(x)  
valueLabel(x) <- value  
  
valueName(x)  
valueName(x) <- value  
  
valueUnit(x)  
valueUnit(x) <- value
```

**Arguments**

x	an object
value	for valueTags<- a named list of value tags. a character vector of length 1, or NULL.

## Details

These functions store the a short name of for the contents, a longer label that is useful for display, and the units of the contents that is useful for display.

valueTag is an accessor, and valueTag<- is a replacement function for all of the value's information.

valueName is an accessor, and valueName<- is a replacement function for the value's name. This name is used when a plot or a latex table needs a short name and the variable name is not useful.

valueLabel is an accessor, and valueLabel<- is a replacement function for the value's label. The label is used in a plots or latex tables when they need a descriptive name.

valueUnit is an accessor, and valueUnit<- is a replacement function for the value's unit. The unit is used to add unit information to the R output.

## Value

valueTag returns NULL or a named list with each of the named values name, label, unit set if they exists in the object.

For valueTag<- returns list

For valueName, valueLabel, and valueUnit returns NULL or character vector of length 1.

For valueName<-, valueLabel<-, and valueUnit returns value

## Author(s)

Charles Dupont

## See Also

[names](#), [attributes](#)

## Examples

```
age <- c(21,65,43)
y <- 1:3
valueLabel(age) <- "Age in Years"
plot(age, y, xlab=valueLabel(age))
```

```
x1 <- 1:10
x2 <- 10:1
valueLabel(x2) <- 'Label for x2'
valueUnit(x2) <- 'mmHg'
x2
x2[1:5]
dframe <- data.frame(x1, x2)
Label(dframe)
```

```
##In these examples of llist, note that labels are printed after
##variable names, because of print.labelled
```

```
a <- 1:3
b <- 4:6
valueLabel(b) <- 'B Label'
```

---

varclus	<i>Variable Clustering</i>
---------	----------------------------

---

## Description

Does a hierarchical cluster analysis on variables, using the Hoeffding D statistic, squared Pearson or Spearman correlations, or proportion of observations for which two variables are both positive as similarity measures. Variable clustering is used for assessing collinearity, redundancy, and for separating variables into clusters that can be scored as a single variable, thus resulting in data reduction. For computing any of the three similarity measures, pairwise deletion of NAs is done. The clustering is done by `hclust()`. A small function `naclus` is also provided which depicts similarities in which observations are missing for variables in a data frame. The similarity measure is the fraction of NAs in common between any two variables. The diagonals of this `sim` matrix are the fraction of NAs in each variable by itself. `naclus` also computes `na.per.obs`, the number of missing variables in each observation, and `mean.na`, a vector whose *i*th element is the mean number of missing variables other than variable *i*, for observations in which variable *i* is missing. The `naplot` function makes several plots (see the `which` argument).

So as to not generate too many dummy variables for multi-valued character or categorical predictors, `varclus` will automatically combine infrequent cells of such variables using `combine.levels`.

`plotMultSim` plots multiple similarity matrices, with the similarity measure being on the x-axis of each subplot.

`na.pattern` prints a frequency table of all combinations of missingness for multiple variables. If there are 3 variables, a frequency table entry labeled 110 corresponds to the number of observations for which the first and second variables were missing but the third variable was not missing.

## Usage

```
varclus(x, similarity=c("spearman", "pearson", "hoeffding", "bothpos", "ccbothpos"),
        type=c("data.matrix", "similarity.matrix"),
        method="complete",
        data=NULL, subset=NULL, na.action=na.retain,
        trans=c("square", "abs", "none"), ...)
## S3 method for class 'varclus'
print(x, abbrev=FALSE, ...)
## S3 method for class 'varclus'
plot(x, ylab, abbrev=FALSE, legend.=FALSE, loc, maxlen, labels, ...)

naclus(df, method)
naplot(obj, which=c('all', 'na per var', 'na per obs', 'mean na',
                   'na per var vs mean na'), ...)

plotMultSim(s, x=1:dim(s)[3],
```

```

slim=range(pretty(c(0,max(s,na.rm=TRUE))))),
slimds=FALSE,
add=FALSE, lty=par('lty'), col=par('col'),
lwd=par('lwd'), vname=NULL, h=.5, w=.75, u=.05,
labelx=TRUE, xspace=.35)

```

```
na.pattern(x)
```

### Arguments

x	<p>a formula, a numeric matrix of predictors, or a similarity matrix. If x is a formula, <code>model.matrix</code> is used to convert it to a design matrix. If the formula excludes an intercept (e.g., <math>\sim a + b - 1</math>), the first categorical (factor) variable in the formula will have dummy variables generated for all levels instead of omitting one for the first level. For <code>plot</code> and <code>print</code>, x is an object created by <code>varclus</code>. For <code>na.pattern</code>, x is a data table, data frame, or matrix.</p> <p>For <code>plotMultSim</code>, is a numeric vector specifying the ordered unique values on the x-axis, corresponding to the third dimension of s.</p>
df	a data frame
s	<p>an array of similarity matrices. The third dimension of this array corresponds to different computations of similarities. The first two dimensions come from a single similarity matrix. This is useful for displaying similarity matrices computed by <code>varclus</code>, for example. A use for this might be to show pairwise similarities of variables across time in a longitudinal study (see the example below). If <code>vname</code> is not given, s must have <code>dimnames</code>.</p>
similarity	<p>the default is to use squared Spearman correlation coefficients, which will detect monotonic but nonlinear relationships. You can also specify linear correlation or Hoeffding's (1948) D statistic, which has the advantage of being sensitive to many types of dependence, including highly non-monotonic relationships. For binary data, or data to be made binary, <code>similarity="bothpos"</code> uses as a similarity measure the proportion of observations for which two variables are both positive. <code>similarity="ccbothpos"</code> uses a chance-corrected measure which is the proportion of observations for which both variables are positive minus the product of the two marginal proportions. This difference is expected to be zero under independence. For diagonals, <code>"ccbothpos"</code> still uses the proportion of positives for the single variable. So <code>"ccbothpos"</code> is not really a similarity measure, and clustering is not done. This measure is useful for plotting with <code>plotMultSim</code> (see the last example).</p>
type	<p>if x is not a formula, it may be a data matrix or a similarity matrix. By default, it is assumed to be a data matrix.</p>
method	<p>see <code>hclust</code>. The default, for both <code>varclus</code> and <code>naclus</code>, is <code>"compact"</code> (for R it is <code>"complete"</code>).</p>
data	a data frame, data table, or list
subset	a standard subsetting expression
na.action	<p>These may be specified if x is a formula. The default <code>na.action</code> is <code>na.retain</code>, defined by <code>varclus</code>. This causes all observations to be kept in the model frame, with later pairwise deletion of NAs.</p>



trans	By default, when the similarity measure is based on Pearson's or Spearman's correlation coefficients, the coefficients are squared. Specify <code>trans="abs"</code> to take absolute values or <code>trans="none"</code> to use the coefficients as they stand.
...	for <code>varclus</code> these are optional arguments to pass to the <code>dataframeReduce</code> function. Otherwise, passed to <code>plclust</code> (or to <code>dotchart</code> or <code>dotchart2</code> for <code>naplot</code> ).
ylab	y-axis label. Default is constructed on the basis of similarity.
legend.	set to TRUE to plot a legend defining the abbreviations
loc	a list with elements <code>x</code> and <code>y</code> defining coordinates of the upper left corner of the legend. Default is <code>locator(1)</code> .
maxlen	if a legend is plotted describing abbreviations, original labels longer than <code>maxlen</code> characters are truncated at <code>maxlen</code> .
labels	a vector of character strings containing labels corresponding to columns in the similar matrix, if the column names of that matrix are not to be used
obj	an object created by <code>naclus</code>
which	defaults to "all" meaning to have <code>naplot</code> make 4 separate plots. To make only one of the plots, use <code>which="na per var"</code> (dot chart of fraction of NAs for each variable), <code>"na per obs"</code> (dot chart showing frequency distribution of number of variables having NAs in an observation), <code>"mean na"</code> (dot chart showing mean number of other variables missing when the indicated variable is missing), or <code>"na per var vs mean na"</code> , a scatterplot showing on the x-axis the fraction of NAs in the variable and on the y-axis the mean number of other variables that are NA when the indicated variable is NA.
abbrev	set to TRUE to abbreviate variable names for plotting or printing. Is set to TRUE automatically if <code>legend=TRUE</code> .
slim	2-vector specifying the range of similarity values for scaling the y-axes. By default this is the observed range over all of <code>s</code> .
slimds	set to <code>slimds</code> to TRUE to scale diagonals and off-diagonals separately
add	set to TRUE to add similarities to an existing plot (usually specifying <code>lty</code> or <code>col</code> )
lty, col, lwd	line type, color, or line thickness for <code>plotMultSim</code>
vname	optional vector of variable names, in order, used in <code>s</code>
h	relative height for subplot
w	relative width for subplot
u	relative extra height and width to leave unused inside the subplot. Also used as the space between y-axis tick mark labels and graph border.
labelx	set to FALSE to suppress drawing of labels in the x direction
xspace	amount of space, on a scale of 1:n where n is the number of variables, to set aside for y-axis labels

### Details

`options(contrasts= c("contr.treatment", "contr.poly"))` is issued temporarily by `varclus` to make sure that ordinary dummy variables are generated for factor variables. Pass arguments to the `dataframeReduce` function to remove problematic variables (especially if analyzing all variables in a data frame).

**Value**

for varclus or naclus, a list of class varclus with elements call (containing the calling statement), sim (similarity matrix), n (sample size used if x was not a correlation matrix already - n is a matrix), hclust, the object created by hclust, similarity, and method. naclus also returns the two vectors listed under description, and naplot returns an invisible vector that is the frequency table of the number of missing variables per observation. plotMultSim invisibly returns the limits of similarities used in constructing the y-axes of each subplot. For similarity="ccbothpos" the hclust object is NULL.

na.pattern creates an integer vector of frequencies.

**Side Effects**

plots

**Author(s)**

Frank Harrell  
Department of Biostatistics, Vanderbilt University  
<fh@fharrell.com>

**References**

Sarle, WS: The VARCLUS Procedure. SAS/STAT User's Guide, 4th Edition, 1990. Cary NC: SAS Institute, Inc.

Hoeffding W. (1948): A non-parametric test of independence. Ann Math Stat 19:546–57.

**See Also**

[hclust](#), [plclust](#), [hoeffd](#), [rcorr](#), [cor](#), [model.matrix](#), [locator](#), [na.pattern](#), [cut2](#), [combine.levels](#)

**Examples**

```
set.seed(1)
x1 <- rnorm(200)
x2 <- rnorm(200)
x3 <- x1 + x2 + rnorm(200)
x4 <- x2 + rnorm(200)
x <- cbind(x1,x2,x3,x4)
v <- varclus(x, similarity="spear") # spearman is the default anyway
v # invokes print.varclus
print(round(v$sim,2))
plot(v)

# Convert the dendrogram to be horizontal
v <- as.dendrogram(v$hclust)
plot(v, horiz=TRUE, axes=FALSE, xlab=expression(paste('Spearman ', rho^2)))
rh <- seq(0, 1, by=0.1) # re-label x-axis re:similarity not distance
axis(1, at=1 - rh, labels=format(rh))

# plot(varclus(~ age + sys.bp + dias.bp + country - 1), abbrev=TRUE)
```

```

# the -1 causes k dummies to be generated for k countries
# plot(varclus(~ age + factor(disease.code) - 1))
#
#
# use varclus(~., data= fracmiss= maxlevels= minprev=) to analyze all
# "useful" variables - see dataframeReduce for details about arguments

df <- data.frame(a=c(1,2,3),b=c(1,2,3),c=c(1,2,NA),d=c(1,NA,3),
                 e=c(1,NA,3),f=c(NA,NA,NA),g=c(NA,2,3),h=c(NA,NA,3))
par(mfrow=c(2,2))
for(m in c("ward","complete","median")) {
  plot(naclus(df, method=m))
  title(m)
}
naplot(naclus(df))
n <- naclus(df)
plot(n); naplot(n)
na.pattern(df)

# plotMultSim example: Plot proportion of observations
# for which two variables are both positive (diagonals
# show the proportion of observations for which the
# one variable is positive). Chance-correct the
# off-diagonals by subtracting the product of the
# marginal proportions. On each subplot the x-axis
# shows month (0, 4, 8, 12) and there is a separate
# curve for females and males
d <- data.frame(sex=sample(c('female','male'),1000,TRUE),
               month=sample(c(0,4,8,12),1000,TRUE),
               x1=sample(0:1,1000,TRUE),
               x2=sample(0:1,1000,TRUE),
               x3=sample(0:1,1000,TRUE))
s <- array(NA, c(3,3,4))
opar <- par(mar=c(0,0,4.1,0)) # waste less space
for(sx in c('female','male')) {
  for(i in 1:4) {
    mon <- (i-1)*4
    s[,,i] <- varclus(~x1 + x2 + x3, sim='ccbothpos', data=d,
                    subset=d$month==mon & d$sex==sx)$sim
  }
  plotMultSim(s, c(0,4,8,12), vname=c('x1','x2','x3'),
             add=sx=='male', slimds=TRUE,
             lty=1+(sx=='male'))
  # slimds=TRUE causes separate scaling for diagonals and
  # off-diagonals
}
par(opar)

```

**Description**

Easily Retrieve Text Form of Labels/Units

**Usage**

```
vlab(x, name = NULL)
```

**Arguments**

x	a single variable name, unquoted
name	optional character string to use as variable name

**Details**

Uses the same search method as `hlab` returns label and units in a character string with units, if present, in brackets

**Value**

character string

**Author(s)**

Frank Harrell

**See Also**

[hlab\(\)](#)

---

wtd.stats

*Weighted Statistical Estimates*

---

**Description**

These functions compute various weighted versions of standard estimators. In most cases the weights vector is a vector the same length of `x`, containing frequency counts that in effect expand `x` by these counts. weights can also be sampling weights, in which setting `normwt` to `TRUE` will often be appropriate. This results in making weights sum to the length of the non-missing elements in `x`. `normwt=TRUE` thus reflects the fact that the true sample size is the length of the `x` vector and not the sum of the original values of weights (which would be appropriate had `normwt=FALSE`). When weights is all ones, the estimates are all identical to unweighted estimates (unless one of the non-default quantile estimation options is specified to `wtd.quantile`). When missing data have already been deleted for, `x`, weights, and (in the case of `wtd.loess.noiter`) `y`, specifying `na.rm=FALSE` will save computation time. Omitting the weights argument or specifying `NULL` or a zero-length vector will result in the usual unweighted estimates.

`wtd.mean`, `wtd.var`, and `wtd.quantile` compute weighted means, variances, and quantiles, respectively. `wtd.Ecdf` computes a weighted empirical distribution function. `wtd.table` computes a

weighted frequency table (although only one stratification variable is supported at present). `wtd.rank` computes weighted ranks, using mid-ranks for ties. This can be used to obtain Wilcoxon tests and rank correlation coefficients. `wtd.loess.noiter` is a weighted version of `loess.smooth` when no iterations for outlier rejection are desired. This results in especially good smoothing when `y` is binary. `wtd.quantile` removes any observations with zero weight at the beginning. Previously, these were changing the quantile estimates.

`num.denom.setup` is a utility function that allows one to deal with observations containing numbers of events and numbers of trials, by outputting two observations when the number of events and non-events (trials - events) exceed zero. A vector of subscripts is generated that will do the proper duplications of observations, and a new binary variable `y` is created along with usual cell frequencies (weights) for each of the `y=0`, `y=1` cells per observation.

### Usage

```
wtd.mean(x, weights=NULL, normwt="ignored", na.rm=TRUE)
wtd.var(x, weights=NULL, normwt=FALSE, na.rm=TRUE,
        method=c('unbiased', 'ML'))
wtd.quantile(x, weights=NULL, probs=c(0, .25, .5, .75, 1),
             type=c('quantile', '(i-1)/(n-1)', 'i/(n+1)', 'i/n'),
             normwt=FALSE, na.rm=TRUE)
wtd.Ecdf(x, weights=NULL,
         type=c('i/n', '(i-1)/(n-1)', 'i/(n+1)'),
         normwt=FALSE, na.rm=TRUE)
wtd.table(x, weights=NULL, type=c('list', 'table'),
          normwt=FALSE, na.rm=TRUE)
wtd.rank(x, weights=NULL, normwt=FALSE, na.rm=TRUE)
wtd.loess.noiter(x, y, weights=rep(1,n),
                span=2/3, degree=1, cell=.13333,
                type=c('all', 'ordered all', 'evaluate'),
                evaluation=100, na.rm=TRUE)
num.denom.setup(num, denom)
```

### Arguments

<code>x</code>	a numeric vector (may be a character or category or factor vector for <code>wtd.table</code> )
<code>num</code>	vector of numerator frequencies
<code>denom</code>	vector of denominators (numbers of trials)
<code>weights</code>	a numeric vector of weights
<code>normwt</code>	specify <code>normwt=TRUE</code> to make weights sum to <code>length(x)</code> after deletion of NAs. If weights are frequency weights, then <code>normwt</code> should be <code>FALSE</code> , and if weights are normalization (aka reliability) weights, then <code>normwt</code> should be <code>TRUE</code> . In the case of the former, no check is made that weights are valid frequencies.
<code>na.rm</code>	set to <code>FALSE</code> to suppress checking for NAs
<code>method</code>	determines the estimator type; if <code>'unbiased'</code> (the default) then the usual unbiased estimate (using Bessel's correction) is returned, if <code>'ML'</code> then it is the maximum likelihood estimate for a Gaussian distribution. In the case of the latter, the <code>normwt</code> argument has no effect. Uses <code>stats:cov.wt</code> for both methods.

probs	a vector of quantiles to compute. Default is 0 (min), .25, .5, .75, 1 (max).
type	For <code>wtd.quantile</code> , <code>type</code> defaults to <code>quantile</code> to use the same interpolated order statistic method as <code>quantile</code> . Set <code>type</code> to " $(i-1)/(n-1)$ ", " $i/(n+1)$ ", or " $i/n$ " to use the inverse of the empirical distribution function, using, respectively, $(wt - 1)/T$ , $wt/(T+1)$ , or $wt/T$ , where $wt$ is the cumulative weight and $T$ is the total weight (usually total sample size). These three values of <code>type</code> are the possibilities for <code>wtd.Ecdf</code> . For <code>wtd.table</code> the default <code>type</code> is "list", meaning that the function is to return a list containing two vectors: $x$ is the sorted unique values of $x$ and <code>sum.of.weights</code> is the sum of weights for that $x$ . This is the default so that you don't have to convert the names attribute of the result that can be obtained with <code>type="table"</code> to a numeric variable when $x$ was originally numeric. <code>type="table"</code> for <code>wtd.table</code> results in an object that is the same structure as those returned from <code>table</code> . For <code>wtd.loess.noiter</code> the default <code>type</code> is "all", indicating that the function is to return a list containing all the original values of $x$ (including duplicates and without sorting) and the smoothed $y$ values corresponding to them. Set <code>type="ordered all"</code> to sort by $x$ , and <code>type="evaluate"</code> to evaluate the smooth only at evaluation equally spaced points between the observed limits of $x$ .
y	a numeric vector the same length as $x$
span, degree, cell, evaluation	see <code>loess.smooth</code> . The default is linear ( <code>degree=1</code> ) and 100 points to evaluation (if <code>type="evaluate"</code> ).

### Details

The functions correctly combine weights of observations having duplicate values of  $x$  before computing estimates.

When `normwt=FALSE` the weighted variance will not equal the unweighted variance even if the weights are identical. That is because of the subtraction of 1 from the sum of the weights in the denominator of the variance formula. If you want the weighted variance to equal the unweighted variance when weights do not vary, use `normwt=TRUE`. The articles by Gatz and Smith discuss alternative approaches, to arrive at estimators of the standard error of a weighted mean.

`wtd.rank` does not handle NAs as elegantly as `rank` if `weights` is specified.

### Value

`wtd.mean` and `wtd.var` return scalars. `wtd.quantile` returns a vector the same length as `probs`. `wtd.Ecdf` returns a list whose elements  $x$  and `Ecdf` correspond to unique sorted values of  $x$ . If the first CDF estimate is greater than zero, a point  $(\min(x), 0)$  is placed at the beginning of the estimates. See above for `wtd.table`. `wtd.rank` returns a vector the same length as  $x$  (after removal of NAs, depending on `na.rm`). See above for `wtd.loess.noiter`.

### Author(s)

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University School of Medicine  
 <fh@fharrell.com>

Benjamin Tyner  
<btynr@gmail.com>

## References

Research Triangle Institute (1995): SUDAAN User's Manual, Release 6.40, pp. 8-16 to 8-17.

Gatz DF, Smith L (1995): The standard error of a weighted mean concentration–I. Bootstrapping vs other methods. *Atmospheric Env* 11:1185-1193.

Gatz DF, Smith L (1995): The standard error of a weighted mean concentration–II. Estimating confidence intervals. *Atmospheric Env* 29:1195-1200.

[https://en.wikipedia.org/wiki/Weighted\\_arithmetic\\_mean](https://en.wikipedia.org/wiki/Weighted_arithmetic_mean)

## See Also

[mean](#), [var](#), [quantile](#), [table](#), [rank](#), [loess.smooth](#), [lowess](#), [plsmo](#), [Ecdf](#), [somers2](#), [describe](#)

## Examples

```
set.seed(1)
x <- runif(500)
wts <- sample(1:6, 500, TRUE)
std.dev <- sqrt(wtd.var(x, wts))
wtd.quantile(x, wts)
death <- sample(0:1, 500, TRUE)
plot(wtd.loess.noiter(x, death, wts, type='evaluate'))
describe(~x, weights=wts)
# describe uses wtd.mean, wtd.quantile, wtd.table
xg <- cut2(x,g=4)
table(xg)
wtd.table(xg, wts, type='table')

# Here is a method for getting stratified weighted means
y <- runif(500)
g <- function(y) wtd.mean(y[,1],y[,2])
summarize(cbind(y, wts), llist(xg), g, stat.name='y')

# Empirically determine how methods used by wtd.quantile match with
# methods used by quantile, when all weights are unity
set.seed(1)
u <- eval(formals(wtd.quantile)$type)
v <- as.character(1:9)
r <- matrix(0, nrow=length(u), ncol=9, dimnames=list(u,v))

for(n in c(8, 13, 22, 29))
{
  x <- rnorm(n)
  for(i in 1:5) {
    probs <- sort( runif(9))
    for(wtype in u) {
      wq <- wtd.quantile(x, type=wtype, weights=rep(1,length(x)), probs=probs)
      for(qtype in 1:9) {
```

```

      rq <- quantile(x, type=qtype, probs=probs)
      r[wtype, qtype] <- max(r[wtype,qtype], max(abs(wq-rq)))
    }
  }
}
}

r

# Restructure data to generate a dichotomous response variable
# from records containing numbers of events and numbers of trials
num <- c(10,NA,20,0,15) # data are 10/12 NA/999 20/20 0/25 15/35
denom <- c(12,999,20,25,35)
w <- num.denom.setup(num, denom)
w
# attach(my.data.frame[w$subs,])

```

---

xtfrm.labelled

*Auxiliary Function Method for Sorting and Ranking*


---

### Description

An auxiliary function method that is a workaround for bug in the implementation of xtfrm handles inheritance.

### Usage

```
## S3 method for class 'labelled'
xtfrm(x)
```

### Arguments

x any object of class labelled.

### See Also

[xtfrm](#)

---

xy.group

*Mean x vs. function of y in groups of x*


---

### Description

Compute mean x vs. a function of y (e.g. median) by quantile groups of x or by x grouped to have a given minimum number of observations. Deletes NAs in x and y before doing computations.



**Usage**

```
xy.group(x, y, m=150, g, fun=mean, result="list")
```

**Arguments**

x	a vector, may contain NAs
y	a vector of same length as x, may contain NAs
m	number of observations per group
g	number of quantile groups
fun	function of y such as median or mean (the default)
result	"list" (the default), or "matrix"

**Value**

if result="list", a list with components x and y suitable for plotting. if result="matrix", matrix with rows corresponding to x-groups and columns named n, x, and y.

**See Also**

[cut2](#), [cutGn](#), [tapply](#)

**Examples**

```
## Not run:
plot(xy.group(x, y, g=10)) #Plot mean y by deciles of x
xy.group(x, y, m=100, result="matrix") #Print table, 100 obs/group

## End(Not run)
```

---

xYplot

*xyplot and dotplot with Matrix Variables to Plot Error Bars and Bands*


---

**Description**

A utility function `Cbind` returns the first argument as a vector and combines all other arguments into a matrix stored as an attribute called "other". The arguments can be named (e.g., `Cbind(pressure=y, ylow, yhigh)`) or a label attribute may be pre-attached to the first argument. In either case, the name or label of the first argument is stored as an attribute "label" of the object returned by `Cbind`. Storing other vectors as a matrix attribute facilitates plotting error bars, etc., as `trellis` really wants the x- and y-variables to be vectors, not matrices. If a single argument is given to `Cbind` and that argument is a matrix with column dimnames, the first column is taken as the main vector and remaining columns are taken as "other". A subscript method for `Cbind` objects subscripts the other matrix along with the main y vector.

The `xYplot` function is a substitute for `xyplot` that allows for simulated multi-column y. It uses by default the `panel.xYplot` and `prepanel.xYplot` functions to do the actual work. The method argument passed to `panel.xYplot` from `xYplot` allows you to make error bars, the upper-only or

lower-only portions of error bars, alternating lower-only and upper-only bars, bands, or filled bands. `panel.xYplot` decides how to alternate upper and lower bars according to whether the median  $y$  value of the current main data line is above the median  $y$  for all groups of lines or not. If the median is above the overall median, only the upper bar is drawn. For bands (but not 'filled bands'), any number of other columns of  $y$  will be drawn as lines having the same thickness, color, and type as the main data line. If plotting bars, bands, or filled bands and only one additional column is specified for the response variable, that column is taken as the half width of a precision interval for  $y$ , and the lower and upper values are computed automatically as  $y$  plus or minus the value of the additional column variable.

When a `groups` variable is present, `panel.xYplot` will create a function in frame 0 (`.GlobalEnv` in R) called `Key` that when invoked will draw a key describing the groups labels, point symbols, and colors. By default, the key is outside the graph. For S-Plus, if `Key(locator(1))` is specified, the key will appear so that its upper left corner is at the coordinates of the mouse click. For R/Lattice the first two arguments of `Key` ( $x$  and  $y$ ) are fractions of the page, measured from the lower left corner, and the default placement is at  $x=0.05$ ,  $y=0.95$ . For R, an optional argument to `sKey`, `other`, may contain a list of arguments to pass to `draw.key` (see [xyplot](#) for a list of possible arguments, under the key option).

When `method="quantile"` is specified, `xYplot` automatically groups the  $x$  variable into intervals containing a target of  $n_x$  observations each, and within each  $x$  group computes three quantiles of  $y$  and plots these as three lines. The mean  $x$  within each  $x$  group is taken as the  $x$ -coordinate. This will make a useful empirical display for large datasets in which scatterdiagrams are too busy to see patterns of central tendency and variability. You can also specify a general function of a data vector that returns a matrix of statistics for the `method` argument. Arguments can be passed to that function via a list `methodArgs`. The statistic in the first column should be the measure of central tendency. Examples of useful `method` functions are those listed under the help file for `summary.formula` such as `smean.c1.normal`.

`xYplot` can also produce bubble plots. This is done when `size` is specified to `xYplot`. When `size` is used, a function `sKey` is generated for drawing a key to the character sizes. See the bubble plot example. `size` can also specify a vector where the first character of each observation is used as the plotting symbol, if `rangeCex` is set to a single `cex` value. An optional argument to `sKey`, `other`, may contain a list of arguments to pass to `draw.key` (see [xyplot](#) for a list of possible arguments, under the key option). See the bubble plot example.

`Dotplot` is a substitute for `dotplot` allowing for a matrix  $x$ -variable, automatic superpositioning when `groups` is present, and creation of a `Key` function. When the  $x$ -variable (created by `Cbind` to simulate a matrix) contains a total of 3 columns, the first column specifies where the dot is positioned, and the last 2 columns specify starting and ending points for intervals. The intervals are shown using line type, width, and color from the `trellis.plot.line` list. By default, you will usually see a darker line segment for the low and high values, with the dotted reference line elsewhere. A good choice of the `pch` argument for such plots is 3 (plus sign) if you want to emphasize the interval more than the point estimate. When the  $x$ -variable contains a total of 5 columns, the 2nd and 5th columns are treated as the 2nd and 3rd are treated above, and the 3rd and 4th columns define an inner line segment that will have twice the thickness of the outer segments. In addition, tick marks separate the outer and inner segments. This type of display (an example of which appeared in *The Elements of Graphing Data* by Cleveland) is very suitable for displaying two confidence levels (e.g., 0.9 and 0.99) or the 0.05, 0.25, 0.75, 0.95 sample quantiles, for example. For this display, the central point displays well with a default circle symbol.

`setTrellis` sets nice defaults for Trellis graphics, assuming that the graphics device has already

been opened if using postscript, etc. By default, it sets panel strips to blank and reference dot lines to thickness 1 instead of the Trellis default of 2.

`numericScale` is a utility function that facilitates using `xYplot` to plot variables that are not considered to be numeric but which can readily be converted to numeric using `as.numeric()`. `numericScale` by default will keep the name of the input variable as a label attribute for the new numeric variable.

## Usage

```
Cbind(...)
```

```
xYplot(formula, data = sys.frame(sys.parent()), groups,
       subset, xlab=NULL, ylab=NULL, ylim=NULL,
       panel=panel.xYplot, prepanel=prepanel.xYplot, scales=NULL,
       minor.ticks=NULL, sub=NULL, ...)
```

```
panel.xYplot(x, y, subscripts, groups=NULL,
            type=if(is.function(method) || method=='quantiles')
                'b' else 'p',
            method=c("bars", "bands", "upper bars", "lower bars",
                    "alt bars", "quantiles", "filled bands"),
            methodArgs=NULL, label.curves=TRUE, abline,
            probs=c(.5,.25,.75), nx=NULL,
            cap=0.015, lty.bar=1,
            lwd=plot.line$lwd, lty=plot.line$lty, pch=plot.symbol$pch,
            cex=plot.symbol$cex, font=plot.symbol$font, col=NULL,
            lwd.bands=NULL, lty.bands=NULL, col.bands=NULL,
            minor.ticks=NULL, col.fill=NULL,
            size=NULL, rangeCex=c(.5,3), ...)
```

```
prepanel.xYplot(x, y, ...)
```

```
Dotplot(formula, data = sys.frame(sys.parent()), groups, subset,
        xlab = NULL, ylab = NULL, ylim = NULL,
        panel=panel.Dotplot, prepanel=prepanel.Dotplot,
        scales=NULL, xscale=NULL, ...)
```

```
prepanel.Dotplot(x, y, ...)
```

```
panel.Dotplot(x, y, groups = NULL,
             pch = dot.symbol$pch,
             col = dot.symbol$col, cex = dot.symbol$cex,
             font = dot.symbol$font, abline, ...)
```

```
setTrellis(strip.blank=TRUE, lty.dot.line=2, lwd.dot.line=1)
```

```
numericScale(x, label=NULL, ...)
```

**Arguments**

...	for Cbind ... is any number of additional numeric vectors. Unless you are using Dotplot (which allows for either 2 or 4 "other" variables) or xYplot with method="bands", vectors after the first two are ignored. If drawing bars and only one extra variable is given in ..., upper and lower values are computed as described above. If the second argument to Cbind is a matrix, that matrix is stored in the "other" attribute and arguments after the second are ignored. For bubble plots, name an argument cex. Also can be other arguments to pass to labcurve.
formula	a trellis formula consistent with xyplot or dotplot
x	x-axis variable. For numericScale x is any vector such as as.numeric(x) returns a numeric vector suitable for x- or y-coordinates.
y	a vector, or an object created by Cbind for xYplot. y represents the main variable to plot, i.e., the variable used to draw the main lines. For Dotplot the first argument to Cbind will be the main x-axis variable.
data, subset, ylim, subscripts, groups, type, scales, panel, prepanel, xlab, ylab	see trellis.args. xlab and ylab get default values from "label" attributes.
xscale	allows one to use the default scales but specify only the x component of it for Dotplot
method	defaults to "bars" to draw error-bar type plots. See meaning of other values above. method can be a function. Specifying method=quantile, methodArgs=list(probs=c(.5, .25, .75)) is the same as specifying method="quantile" without specifying probs.
methodArgs	a list containing optional arguments to be passed to the function specified in method
label.curves	set to FALSE to suppress invocation of labcurve to label primary curves where they are most separated or to draw a legend in an empty spot on the panel. You can also set label.curves to a list of options to pass to labcurve. These options can also be passed as ... to xYplot. See the examples below.
abline	a list of arguments to pass to panel.abline for each panel, e.g. list(a=0, b=1, col=3) to draw the line of identity using color 3. To make multiple calls to panel.abline, pass a list of unnamed lists as abline, e.g., abline=list(list(h=0), list(v=1)).
probs	a vector of three quantiles with the quantile corresponding to the central line listed first. By default probs=c(.5, .25, .75). You can also specify probs through methodArgs=list(probs=...).
nx	number of target observations for each x group (see cut2 m argument). nx defaults to the minimum of 40 and the number of points in the current stratum divided by 4. Set nx=FALSE or nx=0 if x is already discrete and requires no grouping.
cap	the half-width of horizontal end pieces for error bars, as a fraction of the length of the x-axis
lty.bar	line type for bars

<code>lwd, lty, pch, cex, font, col</code>	see <code>trellis.args</code> . These are vectors when <code>groups</code> is present, and the order of their elements corresponds to the different groups, regardless of how many bands or bars are drawn. If you don't specify <code>lty.bands</code> , for example, all band lines within each group will have the same <code>lty</code> .
<code>lty.bands, lwd.bands, col.bands</code>	used to allow <code>lty, lwd, col</code> to vary across the different band lines for different groups. These parameters are vectors or lists whose elements correspond to the added band lines (i.e., they ignore the central line, whose line characteristics are defined by <code>lty, lwd, col</code> ). For example, suppose that 4 lines are drawn in addition to the central line. Specifying <code>lwd.bands=1:4</code> will cause line widths of 1:4 to be used for every group, regardless of the value of <code>lwd</code> . To vary characteristics over the groups use e.g. <code>lwd.bands=list(rep(1,4), rep(2,4))</code> or <code>list(c(1,2,1,2), c(3,4,3,4))</code> .
<code>minor.ticks</code>	a list with elements <code>at</code> and <code>labels</code> specifying positions and labels for minor tick marks to be used on the x-axis of each panel, if any.
<code>sub</code>	an optional subtitle
<code>col.fill</code>	used to override default colors used for the bands in <code>method='filled bands'</code> . This is a vector when <code>groups</code> is present, and the order of the elements corresponds to the different groups, regardless of how many bands are drawn. The default colors for 'filled bands' are pastel colors matching the default colors <code>superpose.line\$col</code> ( <code>plot.line\$col</code> )
<code>size</code>	a vector the same length as <code>x</code> giving a variable whose values are a linear function of the size of the symbol drawn. This is used for example for bubble plots.
<code>rangeCex</code>	a vector of two values specifying the range in character sizes to use for the size variable (lowest first, highest second). <code>size</code> values are linearly translated to this range, based on the observed range of <code>size</code> when <code>x</code> and <code>y</code> coordinates are not missing. Specify a single numeric <code>cex</code> value for <code>rangeCex</code> to use the first character of each observations's size as the plotting symbol.
<code>strip.blank</code>	set to <code>FALSE</code> to not make the panel strip backgrounds blank
<code>lty.dot.line</code>	line type for dot plot reference lines (default = 1 for dotted; use 2 for dotted)
<code>lwd.dot.line</code>	line thickness for reference lines for dot plots (default = 1)
<code>label</code>	a scalar character string to be used as a variable label after <code>numericScale</code> converts the variable to numeric form

### Details

Unlike `xyplot`, `xYplot` senses the presence of a `groups` variable and automatically invokes `panel.superpose` instead of `panel.xyplot`. The same is true for `Dotplot` vs. `dotplot`.

### Value

`Cbind` returns a matrix with attributes. Other functions return standard `trellis` results.

### Side Effects

`plots`, and `panel.xYplot` may create temporary `Key` and `sKey` functions in the session frame.

**Author(s)**

Frank Harrell  
 Department of Biostatistics  
 Vanderbilt University  
 <fh@fharrell.com>  
 Madeline Bauer  
 Department of Infectious Diseases  
 University of Southern California School of Medicine  
 <mbauer@usc.edu>

**See Also**

[xyplot](#), [panel.xyplot](#), [summarize](#), [label](#), [labcurve](#), [errbar](#), [dotplot](#), [reShape](#), [cut2](#), [panel.abline](#)

**Examples**

```
# Plot 6 smooth functions. Superpose 3, panel 2.
# Label curves with p=1,2,3 where most separated
d <- expand.grid(x=seq(0,2*pi,length=150), p=1:3, shift=c(0,pi))
xYplot(sin(x+shift)^p ~ x | shift, groups=p, data=d, type='l')
# Use a key instead, use 3 line widths instead of 3 colors
# Put key in most empty portion of each panel
xYplot(sin(x+shift)^p ~ x | shift, groups=p, data=d,
       type='l', keys='lines', lwd=1:3, col=1)
# Instead of implicitly using labcurve(), put a
# single key outside of panels at lower left corner
xYplot(sin(x+shift)^p ~ x | shift, groups=p, data=d,
       type='l', label.curves=FALSE, lwd=1:3, col=1, lty=1:3)
Key()

# Bubble plots
x <- y <- 1:8
x[2] <- NA
units(x) <- 'cm^2'
z <- 101:108
p <- factor(rep(c('a','b'),4))
g <- c(rep(1,7),2)
data.frame(p, x, y, z, g)
xYplot(y ~ x | p, groups=g, size=z)
Key(other=list(title='g', cex.title=1.2)) # draw key for colors
sKey(.2,.85,other=list(title='Z Values', cex.title=1.2))
# draw key for character sizes

# Show the median and quartiles of height given age, stratified
# by sex and race. Draws 2 sets (male, female) of 3 lines per panel.
# xYplot(height ~ age | race, groups=sex, method='quantiles')
```

```
# Examples of plotting raw data
dfr <- expand.grid(month=1:12, continent=c('Europe','USA'),
                  sex=c('female','male'))
```

```

set.seed(1)
dfr <- upData(dfr,
              y=month/10 + 1*(sex=='female') + 2*(continent=='Europe') +
              runif(48,-.15,.15),
              lower=y - runif(48,.05,.15),
              upper=y + runif(48,.05,.15))

xYplot(Cbind(y,lower,upper) ~ month,subset=sex=='male' & continent=='USA',
       data=dfr)
xYplot(Cbind(y,lower,upper) ~ month|continent, subset=sex=='male',data=dfr)
xYplot(Cbind(y,lower,upper) ~ month|continent, groups=sex, data=dfr); Key()
# add ,label.curves=FALSE to suppress use of labcurve to label curves where
# farthest apart

xYplot(Cbind(y,lower,upper) ~ month,groups=sex,
       subset=continent=='Europe', data=dfr)
xYplot(Cbind(y,lower,upper) ~ month,groups=sex, type='b',
       subset=continent=='Europe', keys='lines',
       data=dfr)
# keys='lines' causes labcurve to draw a legend where the panel is most empty

xYplot(Cbind(y,lower,upper) ~ month,groups=sex, type='b', data=dfr,
       subset=continent=='Europe',method='bands')
xYplot(Cbind(y,lower,upper) ~ month,groups=sex, type='b', data=dfr,
       subset=continent=='Europe',method='upper')

label(dfr$y) <- 'Quality of Life Score'
# label is in Hmisc library = attr(y,'label') <- 'Quality\dots'; will be
# y-axis label
# can also specify Cbind('Quality of Life Score'=y,lower,upper)
xYplot(Cbind(y,lower,upper) ~ month, groups=sex,
       subset=continent=='Europe', method='alt bars',
       offset=grid::unit(.1,'inches'), type='b', data=dfr)
# offset passed to labcurve to label .4 y units away from curve
# for R (using grid/lattice), offset is specified using the grid
# unit function, e.g., offset=grid::unit(.4,'native') or
# offset=grid::unit(.1,'inches') or grid::unit(.05,'npc')

# The following example uses the summarize function in Hmisc to
# compute the median and outer quartiles. The outer quartiles are
# displayed using "error bars"
set.seed(111)
dfr <- expand.grid(month=1:12, year=c(1997,1998), reps=1:100)
month <- dfr$month; year <- dfr$year
y <- abs(month-6.5) + 2*runif(length(month)) + year-1997
s <- summarize(y, llist(month,year), smedian.hilow, conf.int=.5)
xYplot(Cbind(y,Lower,Upper) ~ month, groups=year, data=s,
       keys='lines', method='alt', type='b')

```

```

# Can also do:
s <- summarize(y, llist(month,year), quantile, probs=c(.5,.25,.75),
               stat.name=c('y','Q1','Q3'))
xYplot(Cbind(y, Q1, Q3) ~ month, groups=year, data=s,
        type='b', keys='lines')
# Or:
xYplot(y ~ month, groups=year, keys='lines', nx=FALSE, method='quantile',
        type='b')
# nx=FALSE means to treat month as a discrete variable

# To display means and bootstrapped nonparametric confidence intervals
# use:
s <- summarize(y, llist(month,year), smean.cl.boot)
s
xYplot(Cbind(y, Lower, Upper) ~ month | year, data=s, type='b')
# Can also use Y <- cbind(y, Lower, Upper); xYplot(Cbind(Y) ~ ...)
# Or:
xYplot(y ~ month | year, nx=FALSE, method=smean.cl.boot, type='b')

# This example uses the summarize function in Hmisc to
# compute the median and outer quartiles. The outer quartiles are
# displayed using "filled bands"

s <- summarize(y, llist(month,year), smedian.hilow, conf.int=.5)

# filled bands: default fill = pastel colors matching solid colors
# in superpose.line (this works differently in R)
xYplot ( Cbind ( y, Lower, Upper ) ~ month, groups=year,
         method="filled bands" , data=s, type="l")

# note colors based on levels of selected subgroups, not first two colors
xYplot ( Cbind ( y, Lower, Upper ) ~ month, groups=year,
         method="filled bands" , data=s, type="l",
         subset=(year == 1998 | year == 2000), label.curves=FALSE )

# filled bands using black lines with selected solid colors for fill
xYplot ( Cbind ( y, Lower, Upper ) ~ month, groups=year,
         method="filled bands" , data=s, label.curves=FALSE,
         type="l", col=1, col.fill = 2:3)
Key(.5,.8,col = 2:3) #use fill colors in key

# A good way to check for stable variance of residuals from ols
# xYplot(resid(fit) ~ fitted(fit), method=smean.sdl)
# smean.sdl is defined with summary.formula in Hmisc

```



```

# Plot y vs. a special variable x
# xYplot(y ~ numericScale(x, label='Label for X') | country)
# For this example could omit label= and specify
#   y ~ numericScale(x) | country, xlab='Label for X'

# Here is an example of using xYplot with several options
# to change various Trellis parameters,
# xYplot(y ~ x | z, groups=v, pch=c('1','2','3'),
#       layout=c(3,1),      # 3 panels side by side
#       ylab='Y Label', xlab='X Label',
#       main=list('Main Title', cex=1.5),
#       par.strip.text=list(cex=1.2),
#       strip=function(\dots) strip.default(\dots, style=1),
#       scales=list(alternating=FALSE))

#
# Dotplot examples
#

s <- summarize(y, llist(month,year), smedian.hilow, conf.int=.5)

setTrellis()          # blank conditioning panel backgrounds
Dotplot(month ~ Cbind(y, Lower, Upper) | year, data=s)
# or Cbind(\dots), groups=year, data=s

# Display a 5-number (5-quantile) summary (2 intervals, dot=median)
# Note that summarize produces a matrix for y, and Cbind(y) trusts the
# first column to be the point estimate (here the median)
s <- summarize(y, llist(month,year), quantile,
              probs=c(.5,.05,.25,.75,.95), type='matrix')
Dotplot(month ~ Cbind(y) | year, data=s)
# Use factor(year) to make actual years appear in conditioning title strips

# Plot proportions and their Wilson confidence limits
set.seed(3)
d <- expand.grid(continent=c('USA','Europe'), year=1999:2001,
                reps=1:100)

# Generate binary events from a population probability of 0.2
# of the event, same for all years and continents
d$y <- ifelse(runif(6*100) <= .2, 1, 0)
s <- with(d,
         summarize(y, llist(continent,year),
                   function(y) {
                     n <- sum(!is.na(y))
                     s <- sum(y, na.rm=TRUE)
                     binconf(s, n)
                   }, type='matrix')
        )

```

```

Dotplot(year ~ Cbind(y) | continent, data=s, ylab='Year',
        xlab='Probability')

# Dotplot(z ~ x | g1*g2)
# 2-way conditioning
# Dotplot(z ~ x | g1, groups=g2); Key()
# Key defines symbols for g2

# If the data are organized so that the mean, lower, and upper
# confidence limits are in separate records, the Hmisc reShape
# function is useful for assembling these 3 values as 3 variables
# a single observation, e.g., assuming type has values such as
# c('Mean', 'Lower', 'Upper'):
# a <- reShape(y, id=month, colvar=type)
# This will make a matrix with 3 columns named Mean Lower Upper
# and with 1/3 as many rows as the original data

```

---

yearDays

*Get Number of Days in Year or Month*


---

### Description

Returns the number of days in a specific year or month.

### Usage

```

yearDays(time)

monthDays(time)

```

### Arguments

`time` A POSIXt or Date object describing the month or year in question.

### Author(s)

Charles Dupont

### See Also

[POSIXt](#), [Date](#)

---

`ynbind`*Combine Variables in a Matrix*

---

**Description**

`ynbind` column binds a series of related yes/no variables, allowing for a final argument `label` used to label the panel created for the group. Labels for individual variables are collected into a vector attribute `"labels"` for the result; original variable names are used in place of labels for those variables without labels. A positive response is taken to be `y`, `yes`, `present` (ignoring case) or a logical `TRUE` value. By default, the columns are sorted by ascending order or the overall proportion of positives. A subsetting method is provided for objects of class `"ynbind"`.

`pBlock` creates a matrix similarly labeled, from a general set of variables (without special handling of binaries), and sets to `NA` any observation not in `subset` so that when that block of variables is analyzed it will be only for that subset.

**Usage**

```
ynbind(..., label = deparse(substitute(...)),
       asna = c("unknown", "unspecified"), sort = TRUE)
```

```
pBlock(..., subset=NULL, label = deparse(substitute(...)))
```

**Arguments**

<code>...</code>	a series of vectors
<code>label</code>	a label for the group, to be attached to the resulting matrix as a <code>"label"</code> attribute, used by <a href="#">summaryP</a> .
<code>asna</code>	a vector of character strings specifying levels that are to be treated the same as <code>NA</code> if present
<code>sort</code>	set to <code>FALSE</code> to not sort the columns by their proportions
<code>subset</code>	subset criteria - either a vector of logicals or subscripts

**Value**

a matrix of class `"ynbind"` or `"pBlock"` with `"label"` and `"labels"` attributes. For `"pBlock"`, factor input vectors will have values converted to character.

**Author(s)**

Frank Harrell

**See Also**

[summaryP](#)

**Examples**

```
x1 <- c('yEs', 'no', 'UNKNOWN', NA)
x2 <- c('y', 'n', 'no', 'present')
label(x2) <- 'X2'
X <- ynbind(x1, x2, label='x1-2')
X[1:3,]

pBlock(x1, x2, subset=2:3, label='x1-2')
```

---

%nin%

*Find Matching (or Non-Matching) Elements*

---

**Description**

%nin% is a binary operator, which returns a logical vector indicating if there is a match or not for its left operand. A true vector element indicates no match in left operand, false indicates a match.

**Usage**

```
x %nin% table
```

**Arguments**

x	a vector (numeric, character, factor)
table	a vector (numeric, character, factor), matching the mode of x

**Value**

vector of logical values with length equal to length of x.

**See Also**

[match %in%](#)

**Examples**

```
c('a','b','c') %nin% c('a','b')
```

# Index

- \* **IO**
  - csv.get, 56
  - getZip, 154
  - mdb.get, 226
- \* **NA**
  - aregImpute, 16
- \* **SPSS data file**
  - spss.get, 369
- \* **STATA data file**
  - stata.get, 372
- \* **aggregation**
  - summarize, 379
  - summary.formula, 383
  - summaryM, 398
  - summaryP, 406
  - summaryS, 413
  - xy.group, 472
- \* **algebra**
  - solvet, 357
- \* **aplot**
  - cnvrt.coords, 40
  - labcurve, 186
  - minor.tick, 234
  - pstamp, 286
  - rlegend, 310
  - scat1d, 335
  - show.pch, 348
  - subplot, 377
- \* **apply for matrix**
  - mApply, 220
- \* **apply for vector**
  - mApply, 220
- \* **arith**
  - approxExtrap, 11
- \* **array**
  - print.char.matrix, 281
  - reShape, 307
  - solvet, 357
- \* **attribute**
  - label, 196
  - valueTags, 461
- \* **bootstrap**
  - areg, 12
  - aregImpute, 16
  - bootkm, 30
  - find.matches, 132
  - rm.boot, 311
  - smean.sd, 355
  - transace, 427
  - transcan, 436
- \* **case-control**
  - find.matches, 132
- \* **categorization**
  - cut2, 63
  - ggfreqScatter, 155
- \* **category**
  - binconf, 26
  - biVar, 27
  - bpower, 32
  - bystats, 36
  - cut2, 63
  - dataRep, 74
  - describe, 78
  - mApply, 220
  - mChoice, 222
  - mhgr, 231
  - popower, 274
  - rcorr, 292
  - samplesize.bin, 323
  - simRegOrd, 353
  - summarize, 379
  - summary.formula, 383
  - summaryM, 398
  - summaryP, 406
  - summaryS, 413
  - varclus, 463
  - wtd.stats, 468
  - xy.group, 472

- \* **character**
  - %nin%, 484
  - all.is.numeric, 10
  - capitalize, 38
  - Cs, 55
  - escapeRegex, 106
  - first.word, 136
  - format.df, 137
  - html, 177
  - latex, 203
  - latexTabular, 215
  - latexTherm, 216
  - makeNstr, 220
  - nstr, 251
  - rcspline.restate, 302
  - sedit, 345
  - string.break.line, 375
  - translate, 453
- \* **chron**
  - trunc.POSIXt, 454
  - yearDays, 482
- \* **cluster sampling**
  - deff, 77
- \* **cluster**
  - dataRep, 74
  - varclus, 463
- \* **compressed file**
  - getZip, 154
- \* **concat**
  - makeNstr, 220
- \* **cross-classification**
  - summarize, 379
  - summary.formula, 383
  - summaryP, 406
- \* **data reduction**
  - redun, 305
- \* **datasets**
  - dataRep, 74
- \* **data**
  - contents, 50
  - data.frame.create.modify.check, 65
  - getHdata, 151
  - Save, 334
  - upData, 456
- \* **descriptive statistics**
  - curveRep, 58
- \* **device**
  - showPsfrag, 349
  - tex, 426
- \* **discretization**
  - cut2, 63
  - ggfreqScatter, 155
  - xy.group, 472
- \* **distribution**
  - describe, 78
  - Ecdf, 98
  - hist.data.frame, 163
  - histbackback, 164
  - panel.bpplot, 256
  - rMultinom, 320
  - scatId, 335
  - wtd.stats, 468
- \* **documentation**
  - list.tree, 218
- \* **dplot**
  - approxExtrap, 11
  - cnvrt.coords, 40
  - hist.data.frame, 163
  - histbackback, 164
  - labcurve, 186
  - mgp.axis, 230
  - scatId, 335
  - subplot, 377
- \* **environment**
  - mgp.axis, 230
- \* **epidemiology**
  - find.matches, 132
  - mhgr, 231
- \* **exploratory data analysis**
  - curveRep, 58
- \* **file**
  - csv.get, 56
  - format.df, 137
  - getZip, 154
  - html, 177
  - latex, 203
  - latexTabular, 215
  - latexTherm, 216
  - mdb.get, 226
  - Save, 334
  - spss.get, 369
  - src, 371
  - stata.get, 372
- \* **grouping**
  - bystats, 36
  - cut2, 63

- ggfreqScatter, 155
- summarize, 379
- summary.formula, 383
- summaryM, 398
- summaryP, 406
- summaryS, 413
- wtd.stats, 468
- xy.group, 472
- \* **hplot**
  - bpplot, 34
  - colorFacet, 43
  - curveRep, 58
  - describe, 78
  - dotchart2, 87
  - dotchart3, 89
  - dotchartpl, 93
  - Ecdf, 98
  - errbar, 104
  - event.chart, 112
  - event.convert, 123
  - ggfreqScatter, 155
  - hist.data.frame, 163
  - histbackback, 164
  - histboxp, 165
  - labcurve, 186
  - latexDotchart, 213
  - minor.tick, 234
  - mtitle, 243
  - multLines, 245
  - panel.bpplot, 256
  - plsmo, 269
  - rm.boot, 311
  - scat1d, 335
  - showPsfrag, 349
  - summary.formula, 383
  - summaryM, 398
  - summaryP, 406
  - summaryRc, 411
  - summaryS, 413
  - symbol.freq, 419
  - tex, 426
  - xYplot, 473
- \* **htest**
  - binconf, 26
  - biVar, 27
  - bpower, 32
  - ciapower, 39
  - cpower, 52
  - data.frame.create.modify.check, 65
  - deff, 77
  - find.matches, 132
  - gbayes, 141
  - hoeffd, 175
  - impute, 181
  - mhgr, 231
  - plotCorrPrecision, 265
  - popower, 274
  - rcorr, 292
  - rm.boot, 311
  - samplesize.bin, 323
  - simRegOrd, 353
  - smean.sd, 355
  - spower, 363
  - t.test.cluster, 421
- \* **html**
  - contents, 50
- \* **imputation**
  - aregImpute, 16
- \* **interface**
  - contents, 50
  - data.frame.create.modify.check, 65
  - describe, 78
  - format.df, 137
  - getHdata, 151
  - getRs, 152
  - html, 177
  - knitrSet, 184
  - latex, 203
  - latexTabular, 215
  - latexTherm, 216
  - rcspline.restate, 302
  - sas.get, 324
  - sasxport.get, 331
  - spss.get, 369
  - stata.get, 372
  - summary.formula, 383
  - summaryM, 398
  - sys, 420
  - tabulr, 422
  - units, 455
- \* **iplot**
  - labcurve, 186
  - mgp.axis, 230
  - Misc, 235
- \* **iteration**
  - mApply, 220

- \* **lattice**
  - Ecdf, 98
  - panel.bpplot, 256
  - plsmo, 269
  - reShape, 307
  - showPsfrag, 349
  - tex, 426
  - xYplot, 473
- \* **list**
  - print.char.list, 279
- \* **loess**
  - wtd.stats, 468
- \* **logistic regression model**
  - rcorr.cens, 294
  - rcorpp.cens, 296
  - somers2, 357
- \* **longitudinal data**
  - curveRep, 58
  - reShape, 307
  - rm.boot, 311
- \* **manip**
  - %nin%, 484
  - addMarginal, 9
  - capitalize, 38
  - csv.get, 56
  - data.frame.create.modify.check, 65
  - dataRep, 74
  - discrete, 85
  - escapeRegex, 106
  - first.word, 136
  - format.df, 137
  - html, 177
  - Lag, 201
  - latex, 203
  - latexTabular, 215
  - latexTherm, 216
  - makeNstr, 220
  - mChoice, 222
  - mdb.get, 226
  - nobsY, 250
  - nstr, 251
  - partition, 261
  - prselect, 285
  - reShape, 307
  - sas.get, 324
  - sasxport.get, 331
  - score.binary, 343
  - sedit, 345
  - spss.get, 369
  - stata.get, 372
  - summarize, 379
  - summary.formula, 383
  - summaryM, 398
  - summaryP, 406
  - summaryS, 413
  - trunc.POSIXt, 454
  - upData, 456
  - varclus, 463
  - wtd.stats, 468
- \* **matching**
  - find.matches, 132
- \* **math**
  - find.matches, 132
  - impute, 181
- \* **methods**
  - aregImpute, 16
  - Ecdf, 98
  - format.df, 137
  - html, 177
  - impute, 181
  - latex, 203
  - latexTabular, 215
  - redun, 305
  - transcan, 436
- \* **misc**
  - HmiscOverview, 169
  - label, 196
  - valueTags, 461
  - ynbind, 483
- \* **missing data**
  - aregImpute, 16
- \* **model validation**
  - transace, 427
- \* **models**
  - abs.error.pred, 6
  - areg, 12
  - aregImpute, 16
  - dataRep, 74
  - describe, 78
  - impute, 181
  - na.delete, 246
  - na.detail.response, 247
  - na.keep, 248
  - rcspline.plot, 301
  - redun, 305
  - transcan, 436



- \* **multiple choice**
  - mChoice, 222
- \* **multivariate**
  - areg, 12
  - aregImpute, 16
  - curveRep, 58
  - find.matches, 132
  - pc1, 262
  - redun, 305
  - rm.boot, 311
  - summarize, 379
  - transace, 427
  - transcan, 436
  - varclus, 463
- \* **nonlinear**
  - transace, 427
- \* **nonparametric**
  - biVar, 27
  - bootkm, 30
  - bpplot, 34
  - cut2, 63
  - describe, 78
  - Ecdf, 98
  - hoeffd, 175
  - panel.bpplot, 256
  - plsmo, 269
  - rcorr, 292
  - rcorr.cens, 294
  - rcorrp.cens, 296
  - smean.sd, 355
  - somers2, 357
  - transace, 427
  - wtd.stats, 468
  - xy.group, 472
- \* **ordinal logistic model**
  - popower, 274
  - simRegOrd, 353
- \* **ordinal response**
  - popower, 274
  - simRegOrd, 353
- \* **overview**
  - data.frame.create.modify.check, 65
  - HmiscOverview, 169
- \* **power**
  - bpower, 32
  - ciapower, 39
  - cpower, 52
  - gbayes, 141
  - popower, 274
  - samplesize.bin, 323
  - simRegOrd, 353
  - spower, 363
- \* **predictive accuracy**
  - abs.error.pred, 6
  - rcorr.cens, 294
  - rcorrp.cens, 296
  - somers2, 357
- \* **predictive mean matching**
  - aregImpute, 16
- \* **print**
  - equalBins, 103
  - format.pval, 140
  - print.char.list, 279
  - print.char.matrix, 281
  - prnz, 284
  - simplifyDims, 352
  - string.bounding.box, 375
  - string.break.line, 375
  - stringDims, 376
- \* **programming**
  - data.frame.create.modify.check, 65
  - escapeRegex, 106
  - Misc, 235
  - src, 371
- \* **proportional odds model**
  - popower, 274
  - simRegOrd, 353
- \* **regression**
  - abs.error.pred, 6
  - areg, 12
  - aregImpute, 16
  - na.detail.response, 247
  - rcorrp.cens, 296
  - rcspline.eval, 299
  - rcspline.plot, 301
  - rcspline.restate, 302
  - redun, 305
  - rm.boot, 311
  - transace, 427
  - transcan, 436
- \* **repeated measures**
  - curveRep, 58
  - reShape, 307
  - rm.boot, 311
- \* **representative curves**
  - curveRep, 58

- \* **robust**
  - abs.error.pred, 6
  - describe, 78
  - GiniMd, 158
  - wtd.stats, 468
- \* **serial data**
  - curveRep, 58
- \* **smooth**
  - areg, 12
  - aregImpute, 16
  - plsmo, 269
  - rCspline.eval, 299
  - redun, 305
  - transace, 427
  - transcan, 436
  - wtd.stats, 468
- \* **stratification**
  - summarize, 379
  - summary.formula, 383
  - summaryM, 398
  - summaryP, 406
  - summaryS, 413
  - xy.group, 472
- \* **string**
  - makeNstr, 220
- \* **study design**
  - bpower, 32
  - ciapower, 39
  - cpower, 52
  - deff, 77
  - gbayes, 141
  - popower, 274
  - samplesize.bin, 323
  - simRegOrd, 353
  - spower, 363
- \* **survival**
  - bootkm, 30
  - ciapower, 39
  - cpower, 52
  - event.chart, 112
  - event.convert, 123
  - event.history, 124
  - rcorr.cens, 294
  - rcorrr.cens, 296
  - spower, 363
- \* **trellis**
  - Ecdf, 98
  - panel.bplot, 256
  - plsmo, 269
  - reShape, 307
  - showPsfrag, 349
  - tex, 426
  - xYplot, 473
- \* **univar**
  - GiniMd, 158
  - hdquantile, 160
- \* **utilities**
  - addMarginal, 9
  - consolidate, 49
  - Cs, 55
  - format.df, 137
  - html, 177
  - label, 196
  - latex, 203
  - latexCheckOptions, 212
  - latexTabular, 215
  - latexTherm, 216
  - Misc, 235
  - nobsY, 250
  - nstr, 251
  - prselect, 285
  - Save, 334
  - src, 371
  - tabulr, 422
  - trunc.POSIXt, 454
  - units, 455
  - valueTags, 461
  - yearDays, 482
  - ynbind, 483
- \* **weighted sampling**
  - wtd.stats, 468
- \* **weights**
  - wtd.stats, 468
- .q (Cs), 55
- [, 86
- [.Cbind (xYplot), 473
- [.describe (describe), 78
- [.discrete (discrete), 85
- [.impute (impute), 181
- [.labelled (label), 196
- [.mChoice (mChoice), 222
- [.pBlock (ynbind), 483
- [.roundN (dataRep), 74
- [.special.miss (sas.get), 324
- [.summary.formula.response (summary.formula), 383

- [.transcan (transcan), 436
- [.ynbind (ynbind), 483
- [<- .discrete (discrete), 85
- [[, 86
- [[.discrete (discrete), 85
- %in%, 484
- %nin%, 484
- abbreviate, 224
- abline, 193
- abs.error.pred, 6
- ace, 14, 428, 431, 432, 434, 436, 449
- addggLayers, 7
- addMarginal, 9
- all.digits (sedit), 345
- all.is.numeric, 10
- any, 344
- apply, 134
- approx, 11, 193, 440, 441, 445, 449
- approxExtrap, 11
- areg, 12, 22, 305, 307, 428
- areg.boot (transace), 427
- aregImpute, 16, 437, 440, 447, 449
- arrGrob (colorFacet), 43
- as.character.mChoice (mChoice), 222
- as.data.frame.labelled (label), 196
- as.discrete (discrete), 85
- as.double.mChoice (mChoice), 222
- as.numeric, 10
- as.vector, 309
- asNumericMatrix, 221
- asNumericMatrix (summarize), 379
- assign, 432, 443, 445
- attach, 66
- attributes, 462
- avas, 428, 431, 432, 434, 449
- axis, 230, 234
- ballocation (bpower), 32
- base::list.files(), 202
- base::qr(), 289
- bezier (labcurve), 186
- binconf, 26, 33
- biVar, 27, 294, 295
- bj, 446
- bootcov, 77, 236, 316, 317
- bootkm, 30
- boxplot, 35
- bpower, 32, 53, 276, 366
- bpplot, 34, 259
- bpplotM, 405, 409
- bpplotM (panel.bpplot), 256
- bpplt, 392, 398, 405
- bpplt (panel.bpplot), 256
- bpplt (panel.bpplot), 256
- bsamsize (bpower), 32
- bwplot, 35
- by, 221, 380
- bystats, 36
- bystats2 (bystats), 36
- cancor, 14, 446, 449
- capitalize, 38
- casefold, 66
- cat, 284
- catTestchisq (summary.formula), 383
- Cbind (xYplot), 473
- ceil (trunc.POSIXt), 454
- character.table (show.pch), 348
- chisq.test, 30, 33, 293
- chiSquare (biVar), 27
- chron, 227, 324
- ciapower, 39, 53, 366
- clara, 58, 59, 61
- cleanup.import, 57, 66, 151, 152, 226, 227, 330, 370, 373
- cleanup.import (upData), 456
- clowess (Misc), 235
- cnvrt.coords, 40, 378
- code.levels (sas.get), 324
- coefficients, 440
- colorFacet, 43, 409
- combine.levels, 30, 44, 64, 66, 293, 463, 466
- combineLabels (label), 196
- combplotp, 45, 224
- completer, 22, 47
- concordance, 295, 298, 358
- confbar (Misc), 235
- consolidate, 49
- consolidate<- (consolidate), 49
- contents, 50, 333
- contents(), 130, 168
- contents.list, 333
- conTestkw (summary.formula), 383
- cor, 7, 266, 293, 466
- cor.test, 266
- coxph, 366
- coxph.fit, 301, 302

- cph, [301](#), [302](#), [366](#), [446](#)
- cpower, [40](#), [52](#), [276](#), [366](#)
- Cs, [55](#), [66](#)
- csv.get, [56](#), [227](#)
- cumcategory (summary.formula), [383](#)
- cumsum, [101](#)
- curveRep, [58](#)
- curveSmooth (curveRep), [58](#)
- cut, [37](#), [64](#)
- cut2, [30](#), [37](#), [63](#), [156](#), [293](#), [380](#), [392](#), [466](#), [473](#), [478](#)
- cutGn, [271](#), [473](#)
- cutGn (cut2), [63](#)
- cutGn(), [253](#), [254](#)
- data.frame, [57](#), [66](#), [227](#), [330](#), [372](#), [373](#), [460](#)
- data.frame.create.modify.check, [65](#)
- data.frame.labelled (label), [196](#)
- data.restore, [152](#)
- data.table::melt(), [228](#)
- datadensity, [66](#), [442](#), [449](#)
- datadensity (scat1d), [335](#)
- dataframeReduce, [307](#), [465](#)
- dataframeReduce (upData), [456](#)
- dataRep, [61](#), [74](#)
- Date, [57](#), [119](#), [120](#), [123](#), [227](#), [372](#), [373](#), [455](#), [460](#), [482](#)
- Dates, [324](#), [333](#)
- DateTimeClasses, [333](#), [455](#)
- deff, [77](#)
- density, [341](#), [415](#)
- describe, [51](#), [66](#), [78](#), [182](#), [200](#), [247](#), [249](#), [330](#), [333](#), [460](#), [471](#)
- detach, [66](#)
- dhistboxp (histboxp), [165](#)
- dimnames, [309](#)
- discrete, [85](#)
- dotchart, [89](#), [90](#), [93](#)
- dotchart2, [87](#), [93](#), [392](#)
- dotchart3, [22](#), [30](#), [89](#), [215](#), [293](#), [405](#)
- dotchartp, [96](#)
- dotchartp (dotchart3), [89](#)
- dotchartpl, [93](#)
- dotchartpl(), [255](#)
- Dotplot (xYplot), [473](#)
- dotplot, [413](#), [478](#)
- download.file, [152](#), [153](#)
- draw.key, [310](#), [311](#)
- drawPlot (labcurve), [186](#)
- dvi (latex), [203](#)
- dvigv (latex), [203](#)
- dvips (latex), [203](#)
- ebpcomp, [97](#)
- Ecdf, [20](#), [22](#), [35](#), [98](#), [259](#), [341](#), [471](#)
- ecdfpM (scat1d), [335](#)
- ecdfSteps, [102](#)
- edit, [66](#)
- equalBins, [103](#)
- errbar, [104](#), [478](#)
- escapeBS (escapeRegex), [106](#)
- escapeRegex, [106](#)
- estSeqMarkovOrd, [107](#)
- estSeqSim, [110](#)
- event.chart, [112](#), [123](#), [127](#)
- event.convert, [123](#)
- event.history, [120](#), [123](#), [124](#)
- expand.grid, [66](#)
- extractlabs, [51](#), [129](#), [200](#)
- extractlabs(), [168](#)
- facet\_wrap, [276](#)
- factor, [66](#), [86](#), [344](#), [432](#), [440](#)
- fImport, [130](#)
- find.matches, [132](#)
- first.word, [136](#)
- fit.mult.impute, [22](#), [252](#)
- fit.mult.impute (transcan), [436](#)
- format, [140](#)
- format.default, [392](#), [405](#)
- format.df, [137](#), [208](#), [211](#), [216](#)
- format.mChoice (mChoice), [222](#)
- format.pval, [140](#), [140](#)
- format.special.miss (sas.get), [324](#)
- formatdescribeSingle (describe), [78](#)
- formula, [392](#), [405](#), [413](#)
- formula.summary.formula.cross (summary.formula), [383](#)
- fread, [57](#)
- Function, [428](#), [432](#), [438](#), [445](#)
- Function (transcan), [436](#)
- Function.areg.boot (transace), [427](#)
- Function.transcan, [304](#)
- gbayes, [141](#)
- gbayes1PowerNP (gbayes), [141](#)
- gbayes2 (gbayes), [141](#)
- gbayesMixPost (gbayes), [141](#)

- gbayesMixPowerNP (gbayes), 141
- gbayesMixPredNoData (gbayes), 141
- gbayesSeqSim, 144, 148
- getabd, 150
- getHdata, 151
- getLatestSource (Misc), 235
- getRs, 152
- getZip, 154
- ggfreqScatter, 155
- ggplot, 409, 449
- ggplot.summaryP (summaryP), 406
- ggplot.transace (transace), 427
- ggplot.transcan (transcan), 436
- ggplot2::labs, 169
- ggplot2::labs(), 169
- ggplotlyr, 157
- GiniMd, 83, 158
- Glm, 446
- glm, 440, 446, 447
- Gls, 446
- Gompertz2 (spower), 363
- grep, 106, 347
- grType (Misc), 235
  
- hashCheck, 159
- hclust, 466
- hdquantile, 160
- hidingTOC, 161
- hist, 163, 165, 341
- hist.data.frame, 66, 163, 341
- histbackback, 164
- histboxp, 165, 341
- histboxPM (histboxp), 165
- histogram, 165, 341
- histSpike, 101, 167
- histSpike (scat1d), 335
- histSpikeg, 269, 271
- histSpikeg (scat1d), 335
- hlab, 51, 167, 200
- hlab(), 130, 169, 468
- hlabs, 168
- hlabs(), 168
- Hmisc.Overview (HmiscOverview), 169
- HmiscOverview, 169
- hoeffd, 175, 293, 466
- html, 51, 177, 211
- html.contents.data.frame (contents), 50
- html.describe (describe), 78
- html.summaryM (summaryM), 398
  
- htmlSN (latex), 203
- htmlSpecialType (Misc), 235
- htmltabv, 180
- htmlTranslate (latex), 203
- htmlVerbatim (html), 177
  
- impactPO, 276
- improveProb (rcorrp.cens), 296
- impute, 30, 181, 293, 438, 443, 449
- impute.transcan, 17, 182
- impute.transcan (transcan), 436
- inmChoice (mChoice), 222
- inmChoicelike (mChoice), 222
- interaction, 37
- intMarkovOrd, 182
- inverseFunction, 13
- inverseFunction (Misc), 235
- invertTabulated (transcan), 436
- is.discrete (discrete), 85
- is.imputed (impute), 181
- is.mChoice (mChoice), 222
- is.na<- .discrete (discrete), 85
- is.special.miss (sas.get), 324
  
- james.stein (Misc), 235
- jitter, 341, 441
- jitter2 (scat1d), 335
  
- keepHattrib (Misc), 235
- Key (legendfunctions), 218
- Key2 (legendfunctions), 218
- km.quick (Misc), 235
- knit, 186
- knitrSet, 184
  
- labcurve, 101, 186, 265, 271, 365, 366, 478
- Label (label), 196
- label, 66, 101, 196, 224, 271, 317, 330, 333, 372, 373, 380, 392, 405, 413, 422, 424, 433, 434, 456, 460, 478
- label(), 130, 168, 228
- Label.data.frame (label), 196
- label.data.frame (label), 196
- label.default (label), 196
- label.Surv (label), 196
- label<- (label), 196
- labelLatex, 422
- labelLatex (label), 196
- labelPlotmath (label), 196

- Lag, [201](#)
- lag, [202](#)
- lapply, [83](#), [221](#)
- largest.empty (labcurve), [186](#)
- latestFile, [202](#)
- latex, [37](#), [83](#), [139](#), [179](#), [203](#), [213](#), [304](#), [392](#), [405](#), [424](#)
- latex.bystats (bystats), [36](#)
- latex.bystats2 (bystats), [36](#)
- latex.default, [216](#)
- latex.describe (describe), [78](#)
- latex.summary.formula.cross (summary.formula), [383](#)
- latex.summary.formula.response (summary.formula), [383](#)
- latex.summary.formula.reverse (summary.formula), [383](#)
- latex.summaryM (summaryM), [398](#)
- latex.summaryP (summaryP), [406](#)
- latexBuild (Misc), [235](#)
- latexCheckOptions, [212](#)
- latexDotchart, [213](#)
- latexNeedle (latexTherm), [216](#)
- latexSN (latex), [203](#)
- latexTabular, [215](#)
- latexTherm, [216](#)
- latexTranslate, [392](#), [405](#)
- latexTranslate (latex), [203](#)
- latexVerbatim (latex), [203](#)
- legend, [193](#), [310](#), [311](#)
- legendfunctions, [218](#)
- length<- .discrete (discrete), [85](#)
- list.tree, [218](#)
- llist, [380](#), [392](#)
- llist (label), [196](#)
- lm, [6](#), [7](#), [317](#), [440](#), [446](#)
- lm.fit, [440](#)
- lm.fit.qr.bare, [431](#)
- lm.fit.qr.bare (Misc), [235](#)
- Load (Save), [334](#)
- load, [151](#), [152](#), [335](#)
- locator, [466](#)
- loess.smooth, [471](#)
- Lognorm2 (spower), [363](#)
- logrank, [232](#)
- logrank (spower), [363](#)
- lookup.xport, [333](#)
- lowess, [58](#), [59](#), [271](#), [314](#), [317](#), [341](#), [374](#), [411](#), [471](#)
- lrcum (mhgr), [231](#)
- lrm, [301](#), [302](#), [446](#)
- lrm.fit, [301](#), [302](#)
- lsfit, [449](#)
- makeNstr, [220](#)
- makeSteps (Misc), [235](#)
- mApply, [220](#)
- mapply, [221](#)
- match, [484](#)
- match.mChoice (mChoice), [222](#)
- matchCases (find.matches), [132](#)
- Math.mChoice (mChoice), [222](#)
- matrix, [309](#)
- matrix2dataFrame, [221](#)
- matrix2dataFrame (summarize), [379](#)
- matxv (Misc), [235](#)
- max, [344](#)
- mbarclPanel (summaryS), [413](#)
- mbarclpl (summaryS), [413](#)
- mChoice, [222](#), [392](#), [405](#)
- mdb.get, [226](#)
- Mean (transace), [427](#)
- mean, [471](#)
- medvPanel (summaryS), [413](#)
- medvpl (summaryS), [413](#)
- meltData, [227](#)
- Merge, [229](#)
- mgp.axis, [230](#)
- mgp.axis.labels, [350](#), [427](#)
- mhgr, [231](#)
- mice, [22](#), [437](#), [440](#), [449](#)
- minor.tick, [233](#)
- Misc, [235](#)
- model.frame.default, [83](#), [246](#), [247](#), [249](#)
- model.matrix, [466](#)
- monotone (transace), [427](#)
- monthDays (yearDays), [482](#)
- movStats, [240](#)
- mtext, [244](#)
- mtitle, [243](#)
- multEventChart (poppower), [274](#)
- multinom, [441](#)
- multLines, [245](#)
- na.delete, [83](#), [246](#), [247](#), [249](#), [431](#)
- na.detail.response, [83](#), [246](#), [247](#), [249](#)
- na.include, [182](#)

- na.keep, 83, 246, 248
- na.omit, 246, 247, 249
- na.pattern, 466
- na.pattern (varclus), 463
- na.retain (summary.formula), 383
- naclus, 22, 66, 447, 449
- naclus (varclus), 463
- names, 49, 66, 462
- naplot, 22, 449
- naplot (varclus), 463
- naprint, 83, 246, 247, 249
- naresid, 246, 247, 249
- nchar, 104, 375, 377
- nCoincident, 249
- nFm (tabulr), 422
- nmChoice (mChoice), 222
- nobsY, 250
- nomiss (Misc), 235
- nomogram, 429, 434
- ns, 300, 304
- nstr, 251
- num.denom.setup (wtd.stats), 468
- num.intercepts, 252
- numeric.string (sedit), 345
- numericScale (xYplot), 473
  
- ols, 6, 7, 434, 440, 446
- Ops.mChoice (mChoice), 222
- ordGroupBoot, 253
- ordTestpo (summary.formula), 383
- orm, 236, 252, 445
- outer, 309
- outerText (Misc), 235
  
- page, 66
- pairUpDiff, 254
- panel.abline, 478
- panel.bpplot, 35, 256, 413
- panel.bwplot, 256–259
- panel.Dotplot (xYplot), 473
- panel.Ecdf (Ecdf), 98
- panel.plsmo, 415
- panel.plsmo (plsmo), 269
- panel.superpose, 269, 271
- panel.xYplot (xYplot), 473
- panel.xyplot, 271, 478
- par, 41, 88, 126, 127, 230, 231, 239, 244, 339, 350, 378, 427
- partition, 261
- paste, 220, 251
- pBlock, 409
- pBlock (ynbind), 483
- pc1, 262
- pdf, 350, 427
- pipe, 154
- plclust, 466
- plot, 100, 127, 302, 314, 428
- plot.areg (areg), 12
- plot.areg.boot (transace), 427
- plot.aregImpute (aregImpute), 16
- plot.biVar (biVar), 27
- plot.curveRep (curveRep), 58
- plot.data.frame, 66
- plot.describe, 167
- plot.describe (describe), 78
- plot.drawPlot (labcurve), 186
- plot.gbayes (gbayes), 141
- plot.princmp, 263
- plot.Quantile2 (spower), 363
- plot.rm.boot (rm.boot), 311
- plot.summary.formula.response (summary.formula), 383
- plot.summary.formula.reverse (summary.formula), 383
- plot.summaryM (summaryM), 398
- plot.summaryP (summaryP), 406
- plot.summaryS (summaryS), 413
- plot.transcan (transcan), 436
- plot.varclus (varclus), 463
- plotCorrM, 264
- plotCorrPrecision, 265
- plotlyM, 266
- plotlyParm (Misc), 235
- plotlySave (knitrSet), 184
- plotmath, 90, 168
- plotmathTranslate (label), 196
- plotMultSim (varclus), 463
- plotp (Misc), 235
- plotp.summaryS (summaryS), 413
- plsmo, 269, 341, 411–413, 471
- pMedian, 83, 273
- pngNeedle (latexTherm), 216
- points, 349
- polygon, 127, 315, 317
- pomodm (popower), 274
- popower, 274, 354
- posamsize (popower), 274

- POSIXct, [57](#), [460](#)
- POSIXlt, [455](#)
- POSIXt, [455](#), [482](#)
- postscript, [350](#), [427](#)
- prcomp, [262](#), [449](#)
- predab.resample, [434](#)
- predict, [428](#), [438](#), [442](#), [446](#)
- predict.areg (areg), [12](#)
- predict.areg.boot (transace), [427](#)
- predict.dataRep (dataRep), [74](#)
- predict.transcan (transcan), [436](#)
- prepanel.Dotplot (xYplot), [473](#)
- prepanel.Ecdf (Ecdf), [98](#)
- prepanel.xYplot (xYplot), [473](#)
- princmp, [278](#)
- princmp(), [283](#)
- print, [284](#), [428](#), [443](#)
- print.abs.error.pred (abs.error.pred), [6](#)
- print.areg (areg), [12](#)
- print.areg.boot (transace), [427](#)
- print.aregImpute (aregImpute), [16](#)
- print.arrGrob (colorFacet), [43](#)
- print.biVar (biVar), [27](#)
- print.bystats (bystats), [36](#)
- print.bystats2 (bystats), [36](#)
- print.char.list, [279](#)
- print.char.matrix, [37](#), [281](#), [392](#), [405](#)
- print.contents.data.frame (contents), [50](#)
- print.contents.list (contents), [50](#)
- print.curveRep (curveRep), [58](#)
- print.dataRep (dataRep), [74](#)
- print.describe (describe), [78](#)
- print.dvi (latex), [203](#)
- print.find.matches (find.matches), [132](#)
- print.hoeffd (hoeffd), [175](#)
- print.improveProb (rcorrp.cens), [296](#)
- print.impute (impute), [181](#)
- print.labelled (label), [196](#)
- print.latex (latex), [203](#)
- print.lrcum (mhgr), [231](#)
- print.mChoice (mChoice), [222](#)
- print.mhgr (mhgr), [231](#)
- print.poppower (poppower), [274](#)
- print.posamsize (poppower), [274](#)
- print.predict.dataRep (dataRep), [74](#)
- print.princmp, [282](#)
- print.Quantile2 (spower), [363](#)
- print.rcorr (rcorr), [292](#)
- print.redun (redun), [305](#)
- print.special.miss (sas.get), [324](#)
- print.spower (spower), [363](#)
- print.summary.areg.boot (transace), [427](#)
- print.summary.formula.cross  
(summary.formula), [383](#)
- print.summary.formula.response  
(summary.formula), [383](#)
- print.summary.formula.reverse  
(summary.formula), [383](#)
- print.summary.lm, [140](#)
- print.summary.mChoice (mChoice), [222](#)
- print.summaryM (summaryM), [398](#)
- print.t.test.cluster (t.test.cluster),  
[421](#)
- print.transace (transace), [427](#)
- print.transcan (transcan), [436](#)
- print.varclus (varclus), [463](#)
- printL, [283](#), [284](#)
- printsummaryM (summaryM), [398](#)
- prList (label), [196](#)
- prn (prnz), [284](#)
- prn(), [284](#)
- prnz, [284](#)
- processMI, [449](#)
- propsP0 (poppower), [274](#)
- propsTrans (poppower), [274](#)
- prselect, [285](#)
- prType (Misc), [235](#)
- ps.options, [350](#), [427](#)
- psm, [445](#), [446](#)
- pstamp, [244](#), [286](#)
- putHcap (label), [196](#)
- putHfig (label), [196](#)
- putKey (labcurve), [186](#)
- putKeyEmpty (labcurve), [186](#)
- qcrypt, [287](#)
- qrxcenter, [288](#)
- Quantile (transace), [427](#)
- quantile, [64](#), [83](#), [161](#), [259](#), [271](#), [471](#)
- Quantile.cph, [31](#)
- Quantile2 (spower), [363](#)
- r2describe, [289](#), [307](#)
- R2Measures, [290](#)
- rank, [358](#), [471](#)
- rbind, [352](#)
- rbinom, [320](#)



- `rcorr`, 176, 266, 292, 466
- `rcorr.cens`, 294, 298, 358
- `rcorr.cens (rcorr.cens)`, 294
- `rcorrp.cens`, 295, 296
- `rCs`, 300, 304, 446
- `rCspline.eval`, 299, 301, 302, 304, 317, 449
- `rCspline.plot`, 301
- `rCspline.restate`, 300, 302
- `rCsplineFunction (rCspline.restate)`, 302
- `read.csv`, 57, 460
- `read.dta`, 372, 373
- `read.spss`, 370
- `read.table`, 66
- `read.xport`, 333
- `redun`, 305
- `reformM (aregImpute)`, 16
- `reLabelled (label)`, 196
- `relevel.labelled (label)`, 196
- `rendHTML (Misc)`, 235
- `rep`, 220, 251
- `replace.substring.wild (sedit)`, 345
- `reShape`, 307, 317, 478
- `reshape`, 309
- `restoreHattrib (Misc)`, 235
- `rlegend`, 93, 310
- `rlegendg (rlegend)`, 310
- `rm.boot`, 311
- `rMultinom`, 320
- `robcov`, 77
- `round`, 76, 455
- `roundN (dataRep)`, 74
- `roundPOSIXt (trunc.POSIXt)`, 454
- `rpart`, 441
- `Rq`, 446
- `rug`, 341
- `runifChanged`, 320
- `runParallel`, 322
  
- `sample`, 182
- `samplesize.bin`, 33, 323
- `sapply`, 221, 239
- `sas.codes (sas.get)`, 324
- `sas.get`, 57, 66, 83, 200, 324, 333, 370, 460
- `sasdsLabels`, 51
- `sasdsLabels (sasxport.get)`, 331
- `sasxport.get`, 50, 331
- `Save`, 334
- `save`, 151, 334, 335
- `scale`, 134
  
- `scale_fill_brewer`, 276
- `scan`, 66
- `scat1d`, 163, 167, 193, 259, 271, 335, 415, 416, 443
- `score.binary`, 343
- `sedit`, 345
- `segments`, 341
- `sepUnitsTrans (Misc)`, 235
- `seqFreq`, 348
- `setTrellis`, 350, 427
- `setTrellis (xYplot)`, 473
- `show.col (show.pch)`, 348
- `show.dvi (latex)`, 203
- `show.latex (latex)`, 203
- `show.pch`, 348
- `showPsfrag`, 349
- `simMarkovOrd`, 350
- `simplifyDims`, 352
- `simPOcuts (popower)`, 274
- `simRegOrd`, 276, 353
- `sKey (legendfunctions)`, 218
- `smean.cl.boot (smean.sd)`, 355
- `smean.cl.normal (smean.sd)`, 355
- `smean.sd`, 355, 392, 415
- `smean.sdl (smean.sd)`, 355
- `smearingEst (transace)`, 427
- `smedian.hilow (smean.sd)`, 355
- `solve`, 357
- `solvet`, 357
- `somers2`, 295, 298, 357, 471
- `soprobMarkovOrd`, 359
- `soprobMarkovOrdM`, 360
- `source`, 371
- `spearman (biVar)`, 27
- `spearman2 (biVar)`, 27
- `spikecomp`, 83, 361
- `split`, 261
- `spower`, 40, 53, 363
- `sprintf`, 423
- `spss.get`, 369
- `src`, 371
- `stat_plsmo`, 269, 271, 341, 373
- `stata.get`, 372
- `stats::ecdf()`, 103
- `stats::nlm()`, 184
- `str`, 219
- `strata`, 392
- `stratify`, 413

- stratify (summary.formula), 383
- strgraphwrap (Misc), 235
- string.bounding.box, 375, 377
- string.break.line, 375
- stringDims, 104, 375, 376
- stripplot, 341
- strptime, 56, 57, 458, 460
- strsplit, 376
- strwrap, 239
- subplot, 41, 377
- substring, 347
- substring.location (sedit), 345
- substring2 (sedit), 345
- substring2<- (sedit), 345
- sum, 344
- summarize, 89, 93, 356, 379, 392, 417, 478
- summary, 83, 392, 417, 428, 437, 443, 446
- summary. areg.boot (transace), 427
- summary.data.frame, 66
- summary.find.matches (find.matches), 132
- summary.formula, 66, 356, 383, 411
- summary.impute (impute), 181
- Summary.mChoice (mChoice), 222
- summary.mChoice (mChoice), 222
- summary.transcan (transcan), 436
- summaryD (dotchart3), 89
- summaryDp (dotchart3), 89
- summaryM, 392, 398, 409, 424
- summaryP, 259, 405, 406, 483
- summaryRc, 411
- summaryS, 413
- supsmu, 271, 301, 302, 313, 317
- Surv, 31, 83, 239, 297, 298, 413
- survfit, 31
- Survival.cph, 31
- survreg, 445
- Sweave, 285
- symbol.freq, 419
- symbols, 378, 419, 420
- sys, 420
- system, 421
- t.test, 422
- t.test.cluster, 421
- table, 76, 83, 101, 309, 341, 471
- table\_formatpct (tabulr), 422
- table\_freq (tabulr), 422
- table\_latexdefs (tabulr), 422
- table\_N (tabulr), 422
- table\_options, 422
- table\_pc (tabulr), 422
- table\_trio (tabulr), 422
- tabular, 422–424
- tabulr, 405, 422, 422
- tapply, 83, 221, 473
- testCharDateTime, 425
- tex, 426
- texi2dvi, 211
- text, 193, 349
- timePOSIXt (sas.get), 324
- title, 104, 244
- tobase64image (Misc), 235
- transace, 427
- transcan, 14, 22, 182, 307, 428, 436
- translate, 37, 453
- trap.rule (Misc), 235
- trellis.device, 350, 427
- trellis.strip.blank (Misc), 235
- trunc.POSIXt, 454
- truncPOSIXt (trunc.POSIXt), 454
- units, 66, 317, 422, 455
- units(), 130, 168
- units<- .default (units), 455
- unix, 244, 421
- unPaste (Misc), 235
- upData, 51, 66, 330, 422, 456
- update, 392, 405
- upFirst, 460
- useOuterStrips, 259, 416
- val.prob, 298
- validate, 428, 434
- validate.ols, 7
- valueLabel (valueTags), 461
- valueLabel<- (valueTags), 461
- valueName (valueTags), 461
- valueName<- (valueTags), 461
- valueTags, 461
- valueTags<- (valueTags), 461
- valueUnit (valueTags), 461
- valueUnit<- (valueTags), 461
- var, 471
- varclus, 30, 176, 293, 307, 463
- vcov, 438, 440
- vcov.default (transcan), 436
- vcov.fit.mult.impute (transcan), 436
- vlab, 467

Weibull2 (spower), [363](#)  
whichClosek (Misc), [235](#)  
whichClosePW (Misc), [235](#)  
whichClosest (Misc), [235](#)  
wtd.Ecdf, [101](#)  
wtd.Ecdf (wtd.stats), [468](#)  
wtd.loess.noiter (wtd.stats), [468](#)  
wtd.mean (wtd.stats), [468](#)  
wtd.quantile (wtd.stats), [468](#)  
wtd.rank, [358](#)  
wtd.rank (wtd.stats), [468](#)  
wtd.stats, [468](#)  
wtd.table (wtd.stats), [468](#)  
wtd.var (wtd.stats), [468](#)

xless (Misc), [235](#)  
xtfrm, [472](#)  
xtfrm.labelled, [472](#)  
xy.group, [472](#)  
xYplot, [60](#), [101](#), [193](#), [473](#)  
xyplot, [60](#), [271](#), [311](#), [474](#), [478](#)

yearDays, [482](#)  
ynbind, [406](#), [409](#), [483](#)