Package 'hutils'

October 13, 2022

Type Package

Title Miscellaneous R Functions and Aliases

Version 1.8.1

Date 2022-04-14

Maintainer Hugh Parsonage <hugh.parsonage@gmail.com>

Description Provides utility functions for, and drawing on, the 'data.table' package. The package also collates useful miscellaneous functions extending base R not available elsewhere. The name is a portmanteau of 'utils' and the author.

BugReports https://github.com/hughparsonage/hutils/issues

URL https://github.com/hughparsonage/hutils,
https://hughparsonage.github.io/hutils/

License GPL-3

Depends R (>= 3.3.0)

Imports data.table, magrittr, stats, utils, fastmatch, grDevices

Suggests testthat (>= 2.1.0), datasets, desc, dplyr, digest, fst, Hmisc, hutilscpp, microbenchmark, knitr, rmarkdown, nycflights13, geosphere, ggplot2, readr, rcheology, rstudioapi, survey, tibble, tidyr, withr (>= 2.0.0)

RoxygenNote 7.1.1

Encoding UTF-8

VignetteBuilder knitr

NeedsCompilation no

Author Hugh Parsonage [aut, cre],
Michael Frasco [ctb],
Ben Hamner [ctb]

Repository CRAN

Date/Publication 2022-04-13 15:12:29 UTC

R topics documented:

`hutils-package` *hutils package*

Description

Provides utility functions for, and drawing on, the 'data.table' package. The package also collates useful miscellaneous functions extending base R not available elsewhere. The name is a portmanteau of 'utils' and the author.

Details

The package attempts to provide lightweight, fast, and stable functions for common operations.

By **lightweight**, I mean in terms of dependencies: we import `package:data.table` and `package:fastmatch` which do require compilation, but in C. Otherwise, all dependencies do not require compilation.

By **fast**, I mean essentially as fast as possible without using compilation.

By **stable**, I mean that unit tests *should not change* unless the major version also changes. To make this completely transparent, tests include the version of their introduction and are guaranteed to not be modified (not even in the sense of adding extra, independent tests) while the major version is `1`. Tests that do not include the version in their filename may be modified from version to version (though this will be avoided).

`ahull` *Maximum area given x and y coordinates*

Description

Present since `hutils` `1.2.0`.

Usage

```
ahull(
  DT,
  x = DT$x,
  y = DT$y,
  minH = 0,
  minW = 0,
  maximize = "area",
  incl_negative = FALSE
)
```

Arguments

`DT, x, y`	Coordinates of a curve containing a rectangle. Either as a list, `DT`, containing columns `x` and `y`.
`minH`	The minimum height of the rectangles.
`minW`	The minimum width of the rectangles.
`maximize`	How the rectangle should be selected. Currently, only `"area"` supported.
`incl_negative`	Should areas below the x-axis be considered?

Value

A `data.table`: The coordinates of a rectangle, from (0, 0), (1, 0), (1, 1), (0, 1), south-west clockwise, that is contained within the area of the chart for positive values only.

Examples

```
ahull(, c(0, 1, 2, 3, 4), c(0, 1, 2, 0, 0))
```

`aliases` *Aliases*

Description

These simple aliases can be useful to avoid operator precedence ambiguity, or to make use of indents from commas within your text editor. The all-caps versions accept single-length (capable of 'short-circuits') logical conditions only.

Neithers and nors are identical except have slightly different short-circuits. `NOR` uses negation once so may be quicker if the first argument is very, very prompt.

Usage

```
AND(x, y)

OR(x, y)

nor(x, y)

neither(x, y)

NOR(x, y)

NEITHER(x, y)

pow()

XOR(x, y)
```

Arguments

`x, y`	Logical conditions.

`all_same_sign` *Determine whether a vector is all of the same sign*

Description

Present since `hutils 1.2.0`.

Usage

```
all_same_sign(x)
```

Arguments

`x`	A numeric vector.

Value

`TRUE` if all elements of `x` have the same sign. Zero is a separate sign from positive and negative. All vectors of length-1 or length-0 return `TRUE`, even if `x` = `NA`, (since although the value is unknown, it must have a unique sign), and non-numeric `x`.

Examples

```
all_same_sign(1:10)
all_same_sign(1:10 - 1)
all_same_sign(0)
all_same_sign(NA)
all_same_sign(c(NA, 1))
all_same_sign("surprise?")
all_same_sign(c(0, 0.1 + 0.2 - 0.3))

if (requireNamespace("microbenchmark", quietly = TRUE)) {
  library(microbenchmark)
  microbenchmark(base = length(unique(sign(1:1e5), nmax = 3)) == 1L,
                 all_same_sign(1:1e5))
}
# Unit: microseconds
#                   expr  min   lq mean median   uq  max neval cld
#                   base 2012 2040 2322   2047 2063 9324   100   b
# all_same_sign(1:1e+05)   86   86   94     89   93  290   100  a
```

`any_grepl` *Does the pattern appear anywhere?*

Description

Shortcut for `any(grepl(...))`, mostly for consistency.

Usage

```
any_grepl(
  x,
  pattern,
  perl = TRUE,
  ignore.case = FALSE,
  fixed = FALSE,
  quiet = FALSE
)
```

Arguments

`x`	A character vector.
`pattern, perl, ignore.case, fixed`	As in `grep`.
`quiet`	(logical, default: `FALSE`) If `TRUE`, silences any messages.

Details

From version `v 1.4.0`, `any_grepl(a, bb)` will be internally reversed to `any_grepl(bb, a)` if `length(bb) > 1` and `length(a) == 1`.

Examples

```
any_grepl(c("A_D_E", "K0j"), "[a-z]")
```

---

auc *AUC*

Description

Returns the area under the curve ("AUC") of a receiver-operating characteristic curve for the given predicted and actual values.

Usage

```
auc(actual, pred)
```

Arguments

`actual`	Logical vector: `TRUE` for positive class. If not a logical vector, the result is interpreted as one if safe to do so, *viz.* if `actual` contains precisely two unique values and is either a numeric vector, an ordered factor, or the unique values are `FALSE` and `TRUE` (case-insensitively). Anything else is an error.
`pred`	Numeric (double) vector the same length as `actual` giving the predicted probability of `TRUE`. Must be a numeric vector the same length as `actual`.

Author(s)

Source

Source code based on `Metrics::auc` from Ben Hamner and Michael Frasco and Erin LeDell from the Metrics package.

`average_bearing` *Average of bearings*

Description

Average of bearings

Usage

```
average_bearing(theta1, theta2, average_of_opposite = NULL)

average_bearing_n(thetas)
```

Arguments

`theta1, theta2`	Bearings, expressed in degrees.
`average_of_opposite`	The average of opposing bearings (e.g. average of north and south) is not well-defined. If `NULL`, the result for opposing vectors is undefined; if `"right"`, returns `theta1 + 90`; if `"left"` then `theta2 + 90`. Can also be a single numeric to provide a specific value when the vectors point in opposite directions.
`thetas`	A vector of bearings.

Value

For ‘average_bearing‘, the bearing bisecting the two bearings.

For ‘average_bearing_n‘, the average bearing of the bearing.

Examples

```
average_bearing(0, 90)
average_bearing(0, 270)
average_bearing(90, 180)

average_bearing(0, 180)
average_bearing(0, 180, average_of_opposite = 3)
average_bearing(0, 180, average_of_opposite = "left")

average_bearing_n(1:179)
```

`bearing` *Bearing calculations*

Description

Bearing calculations

Usage

```
bearing(lat_orig, lon_orig, lat_dest, lon_dest)

compass2bearing(compass)

easterly_component(compass)

northerly_component(compass)
```

Arguments

`lat_orig, lon_orig, lat_dest, lon_dest`	Latitude and longitude of origin and destination.
`compass`	A character vector of compass rose points, such as `c("NW", "E", "SSW")`.

Value

`bearing` An approximate bearing from `_orig` and `_dest`.

`compass2bearing` The bearing encoded by the compass input.

`easterly_component` The easterly component of a unit vector pointing in the direction provided.

Examples

```
bearing(0, 0, 90, 0)
bearing(-35, 151, 51, 0)

compass2bearing("NW")
easterly_component("E")
easterly_component("NW")
```

`coalesce` *Find first non-missing element*

Description

Lightweight version of `dplyr::coalesce`, with all the vices and virtues that come from such an approach. Very similar logic (and timings to `dplyr::coalesce`), though no ability to use quosures etc. One exception is that if `x` does not contain any missing values, it is returned immediately, and ignores `...`. For example, `dplyr::coalesce(1:2, 1:3)` is an error, but `hutils::coalesce(1:2, 1:3)` is not.

Usage

```
coalesce(x, ...)
```

Arguments

`x`	A vector
`...`	Successive vectors whose values will replace the corresponding values in `x` if the value is (still) missing.

Value

`x` with missing values replaced by the first non-missing corresponding elements in `...`. That is, if `...` = `A, B, C` and `x[i]` is missing, then `x[i]` is replaced by `A[i]`. If `x[i]` is still missing (i.e. `A[i]` was itself `NA`), then it is replaced by `B[i]`, `C[i]` until it is no longer missing or the list has been exhausted.

Source

Original source code but obviously inspired by `dplyr::coalesce`.

Examples

```
coalesce(c(1, NA, NA, 4), c(1, 2, NA, NA), c(3, 4, 5, NA))
```

`dev_copy2a4` *Copy device to an A4 PDF*

Description

Simply a wrapper around `dev.copy2pdf`, but without the need to remember that an A4 sheet of paper is 8.27 in by 11.69 in.

Usage

```
dev_copy2a4(filename, ...)
```

Arguments

`filename`	A string giving the name of the PDF file to write to, must end in `.pdf`.
`...`	Other parameters passed to `pdf`.

Value

As in `dev2`.

`dir2` *List many files*

Description

(Windows only) Same as `list.files` but much faster.

Present since `v1.4.0`.

Usage

```
dir2(
  path = ".",
  file_ext = NULL,
  full.names = TRUE,
  recursive = TRUE,
  pattern = NULL,
  fixed = FALSE,
  perl = TRUE && missing(fixed) && !fixed,
  ignore.case = FALSE,
  invert = FALSE,
  .dont_use = FALSE
)
```

Arguments

`path`	A string representing the trunk path to search within.
`file_ext`	A string like '*.txt' or '.csv' to limit the result to files with that extension.
`full.names`	`TRUE` by default.
`recursive`	`TRUE` by default.
`pattern, perl, ignore.case, fixed, invert`	As in `grep` but with different defaults. Used to filter files with extension `file_ext`.
`.dont_use`	Only used for tests to simulate non-Windows systems.

Value

The same as list.files, a character vector of files sought.

drop_col *Drop column or columns*

Description

Drop column or columns

Usage

```
drop_col(DT, var, checkDT = TRUE)

drop_cols(DT, vars, checkDT = TRUE)
```

Arguments

DT	A data.table.
var	Quoted column to drop.
checkDT	Should the function check DT is a data.table?
vars	Character vector of columns to drop. Only the intersection is dropped; if any vars are not in names(DT), no warning is emitted.

Value

DT with specified columns removed.

Examples

```
if (requireNamespace("data.table", quietly = TRUE)) {
  library(data.table)
  DT <- data.table(x = 1, y = 2, z = 3)

  drop_col(DT, "x")
}
```

`drop_colr` *Drop columns whose names match a pattern*

Description

`drop_colr` present since `hutils` `1.0.0`.

`drop_grep` is identical but only present since `hutils` `1.2.0`.

Usage

```
drop_colr(DT, pattern, ..., checkDT = TRUE)
```

Arguments

`DT`	A `data.table`.
`pattern`	A regular expression as in `grepl`.
`...`	Arguments passed to `grepl`.
`checkDT`	If `TRUE` (the default), will error if `DT` is not a `data.table`.

Examples

```
library(data.table)
dt <- data.table(x1 = 1, x2 = 2, y = 3)
drop_grep(dt, "x")
```

`drop_constant_cols` *Drop constant columns*

Description

Drops columns that have only one value in a `data.table`.

Usage

```
drop_constant_cols(DT, copy = FALSE)
```

Arguments

`DT`	A `data.table`.
`copy`	(logical, default: `FALSE`) Whether the `data.table` should be copied before any columns are dropped. If `FALSE`, the default, columns are dropped from `DT` by reference.

Details

If DT is a data.frame that is not a data.table, constant columns are still dropped, but since DT will be copied, copy should be set to TRUE to avoid a warning. If DT is a data.frame and all but one of the columns are constant, a data.frame will still be returned, as opposed to the values of the sole remaining column, which is the default behaviour of base data.frame.

If all columns are constant, drop_constant_cols returns a Null data table if DT is a data.table, but a data frame with 0 columns and nrow(DT) otherwise.

Examples

```
library(data.table)
X <- data.table(x = c(1, 1), y = c(1, 2))
drop_constant_cols(X)
```

---

drop_empty_cols *Drop empty columns*

Description

Removes columns from a data.table where all the values are missing.

Usage

```
drop_empty_cols(DT, copy = FALSE)
```

Arguments

DT	A data.table.
copy	Copies the data.table so the original can be retained. Not applicable if DT is not a data.table. If FALSE, the default, DT itself will be modified.

---

duplicated_rows *Return duplicated rows of data.table*

Description

This function differs from duplicated in that it returns both the duplicate row and the row which has been duplicated. This may prove useful in combination with the by argument for determining whether two observations are identical across more than just the specified columns.

Usage

```
duplicated_rows(
  DT,
  by = names(DT),
  na.rm = FALSE,
  order = TRUE,
  copyDT = TRUE,
  na.last = FALSE
)
```

Arguments

`DT`	A `data.table`.
`by`	Character vector of columns to evaluate duplicates over.
`na.rm`	(logical) Should `NA`s in `by` be removed before returning duplicates? (Default `FALSE`.)
`order`	(logical) Should the result be ordered so that duplicate rows are adjacent? (Default `TRUE`.)
`copyDT`	(logical) Should `DT` be copied prior to detecting duplicates. If `FALSE`, the ordering of `DT` will be changed by reference.
`na.last`	(logical) If `order` is TRUE, should `NA`s be ordered first or last?. Passed to `data.table::setorderv`.

Value

Duplicate rows of `DT` by `by`. For interactive use.

Examples

```

if (requireNamespace("data.table", quietly = TRUE)) {
  library(data.table)

  DT <- data.table(x = rep(1:4, 3),
                   y = rep(1:2, 6),
                   z = rep(1:3, 4))

  # No duplicates
  duplicated_rows(DT)

  # x and y have duplicates
  duplicated_rows(DT, by = c("x", "y"), order = FALSE)

  # By default, the duplicate rows are presented adjacent to each other.
  duplicated_rows(DT, by = c("x", "y"))
}
```

find_pattern_in *Find string pattern in (text) file*

Description

goto_pattern_in present from 1.6.0

Usage

```
find_pattern_in(
  file_contents,
  basedir = ".",
  dir_recursive = TRUE,
  reader = readLines,
  include.comments = FALSE,
  comment.char = NULL,
  use.OS = FALSE,
  file_pattern = "\\.(R|r)(nw|md)?$",
  file_contents_perl = TRUE,
  file_contents_fixed = FALSE,
  file_contents_ignore_case = FALSE,
  file.ext = NULL,
  which_lines = c("first", "all")
)

goto_pattern_in(file_contents, ...)
```

Arguments

`file_contents`	A perl-regular expression as a search query.
`basedir`	The root of the directory tree in which files will be searched recursively.
`dir_recursive`	(logical, default: `TRUE`) Search within subdirectories of `basedir`?
`reader`	A function, akin to `base::readLines`, the default, that accepts a filename and returns a character vector.
`include.comments`	If `FALSE`, the default, comments (i.e. anything after a `\#`) are not searched.
`comment.char`	If `include.comments` is `FALSE`, what character marks a comment character? By default, `NULL`, which sets the correct comment symbol for R and TeX files.
`use.OS`	Use the operating system to determine file list. Only available on Windows. If it fails, a fall-back option (using `dir`) is used.
`file_pattern`	A regular expression passed to `list.files(pattern = file.ext)`. By default, `"\.(R|r)(nw|md)?$"`, i.e. all R and Sweave files. (Does not have to be a file extension.)
`file_contents_perl`	(logical, default: `TRUE`) Should `file_contents` be interpreted as a `perl` regex?

`file_contents_fixed`	(logical, default: `FALSE`) Should `file_contents` be interpreted as a `fixed` regex?
`file_contents_ignore_case`	(logical, default: `FALSE`) As in `grep`.
`file.ext`	A file extension passed to the operating system if `use.OS` is used.
`which_lines`	One of `"first"` and `"all"`. If `"first"` only the first match in any file is returned in the result; if `"all"`, all matches are.
`...`	Arguments passed to `find_pattern_in`.

Details

For convenience, if `file_contents` appears to be a directory and `basedir` does not, the arguments are swapped, but with a warning.

Value

A `data.table`, showing the matches per file.

`goto_pattern_in` additionally prompts for a row of the returned results. Using the `rstudioapi`, if available, RStudio will jump to the file and line number.

`fst_columns` *Utilities for ‘fst‘ files*

Description

Utilities for ‘fst‘ files

Usage

```
fst_columns(file.fst)

fst_nrow(file.fst)
```

Arguments

`file.fst`	Path to file.

Value

Various outputs:

`fst_columns` Returns the names of the columns in `file.fst`.

`fst_nrow` Returns the number of rows in `file.fst`.

`generate_LaTeX_manual` *Generate LaTeX manual of installed package*

Description

Generate LaTeX manual of installed package

Usage

```
generate_LaTeX_manual(pkg, launch = TRUE)
```

Arguments

`pkg`	Quoted package name (must be installed).
`launch`	Should the PDF created be launched using the viewer (`TRUE` by default)?

Value

See `system`. Called for its side-effect: creates a PDF in the current working directory. Requires a TeX distribution.

Source

`https://stackoverflow.com/a/30608000/1664978`

`haversine_distance` *Distance between two points on the Earth*

Description

Distance between two points on the Earth

Usage

```
haversine_distance(lat1, lon1, lat2, lon2)
```

Arguments

`lat1, lon1, lat2, lon2`	That latitudes and longitudes of the two points.

Details

This is reasonably accurate for distances in the order of 1 to 1000 km.

Value

The distance in kilometres between the two points.

Examples

```
# Distance from YMEL to YSSY
haversine_distance(-37 - 40/60, 144 + 50/60, -33 - 56/60, 151 + 10/60)
```

`if_else` *Vectorized if*

Description

Lightweight `dplyr::if_else` with the virtues and vices that come from such an approach. Attempts to replicate `dplyr::if_else` but written in base R for faster compile time. `hutils::if_else` should be faster than `dplyr::if_else` . . . when it works, but will not work on lists or on factors. Additional attributes may be dropped.

Usage

```
if_else(condition, true, false, missing = NULL)
```

Arguments

`condition`	Logical vector.
`true, false`	Where condition is `TRUE`/`FALSE`, use the corresponding`true`/`no` value. They must have the same `typeof` as each other and be the same length as `condition` or length-one.
`missing`	If condition is `NA`, use the corresponding `na` value. Like`true` and`false`, must be of the same type and have the same length as condition, unless it has length one.

Details

If the result is expected to be a factor then the conditions for type safety are strict and may be made stricter in future.

Value

Where `condition` is `TRUE`, the corresponding value in `true`; where `condition` is `FALSE`, the corresponding value in `false`. Where `condition` is `NA`, then the corresponding value in `na` – unless `na` is `NULL` (the default) in which case the value will be `NA` (with the same type as `true`.)

Source

Original code but obviously heavily inspired by `https://CRAN.R-project.org/package=dplyr`.

implies *#' Logical implies*

Description

Returns the result of $x \implies y$.

Usage

```
implies(x, y)

x %implies% y
```

Arguments

`x, y`	Logical vectors of the same length.

Value

Logical implies: `TRUE` unless `x` is `TRUE` and `y` is `FALSE`.

`NA` in either `x` or `y` results in `NA` if and only if the result is unknown. In particular `NA %implies% TRUE` is `TRUE` and `FALSE %implies% NA` is `TRUE`.

If `x` or `y` are length-one, the function proceeds as if the length-one vector were recycled to the length of the other.

Examples

```
library(data.table)
CJ(x = c(TRUE,
         FALSE),
   y = c(TRUE,
         FALSE))[, ` x => y` := x %implies% y][]

#>        x     y  x => y
#> 1: FALSE FALSE    TRUE
#> 2: FALSE  TRUE    TRUE
#> 3:  TRUE FALSE   FALSE
#> 4:  TRUE  TRUE    TRUE

# NA results:
#> 5:    NA    NA      NA
#> 6:    NA FALSE      NA
#> 7:    NA  TRUE    TRUE
#> 8: FALSE    NA    TRUE
#> 9:  TRUE    NA      NA
```

`isAttached` *Is a package attached?*

Description

Is a package attached?

Usage

```
isAttached(pkg)
```

Arguments

`pkg`	Either character or unquoted.

Value

`TRUE` if `pkg` is attached.

`isTrueFalse` *Logical assertions*

Description

Logical assertions

Usage

```
isTrueFalse(x)
```

Arguments

`x`	An object whose values are to be checked.

Value

For `isTrueFalse`, `TRUE` if and only if `x` is `TRUE` or `FALSE` identically (perhaps with attributes).

longest_affix *Longest common prefix/suffix*

Description

Longest common prefix/suffix

Usage

```
trim_common_affixes(
  x,
  .x = NULL,
  na.rm = TRUE,
  prefixes = TRUE,
  suffixes = TRUE,
  warn_if_no_prefix = TRUE,
  warn_if_no_suffix = TRUE
)

longest_suffix(x, .x = NULL, na.rm = TRUE, warn_if_no_suffix = TRUE)

longest_prefix(x, .x = NULL, na.rm = TRUE, warn_if_no_prefix = TRUE)
```

Arguments

`x`	A character vector.
`.x`	If `NULL`, the default, ignored. May be used if `x` is known to be free of `NA`s.
`na.rm`	(logical, default: `TRUE`) If `FALSE`, an `NA` in `x` means `""` is the only common affix. If `NA`, the longest prefix/suffix is `NA_character_` (provided `anyNA(x)`). If `anyNA(x) == FALSE` `na.rm` has no effect.
`prefixes`	(logical, default: `TRUE`) If `TRUE`, trim prefixes.
`suffixes`	(logical, default: `TRUE`) If `TRUE`, trim suffixes.
`warn_if_no_prefix, warn_if_no_suffix`	(logical, default: `TRUE`) If `FALSE`, if `x` has no common affixes the warning is suppressed. (If no common prefix/suffix then the common affix returned will be `""` (the empty string).)

Value

The longest common substring in `x` either at the start or end of each string. For `trim_common_affixes` `x` with common prefix and common suffix removed.

Examples

```
longest_prefix(c("totalx", "totaly", "totalz"))
longest_suffix(c("ztotal", "ytotal", "xtotal"))
```

`mean_na` *Proportion of values that are NA.*

Description

Proportion of values that are NA.

Usage

```
mean_na(v)
```

Arguments

`v`	A vector.

Value

A double, `mean(is.na(v))`.

`Mode` *Statistical mode*

Description

Present since `hutils 1.4.0`. The most common element.

Usage

```
Mode(x)
```

Arguments

`x`	A vector for which the mode is desired.

Value

The most common element of `x`.

If the mode is not unique, only one of these values is returned, for simplicity.

If `x` has length zero, `Mode(x) = x`.

mutate_ntile *Add a column of ntiles to a data table*

Description

Add a column of ntiles to a data table

Usage

```
mutate_ntile(
  DT,
  col,
  n,
  weights = NULL,
  by = NULL,
  keyby = NULL,
  new.col = NULL,
  character.only = FALSE,
  overwrite = TRUE,
  check.na = FALSE
)
```

Arguments

`DT`	A `data.table`.
`col`	The column name (quoted or unquoted) for which quantiles are desired.
`n`	A positive integer, the number of groups to split `col`.
`weights`	If `NULL`, the default, use unweighted quantiles. Otherwise, a string designating the column that is passed to `weighted_ntile`.
`by, keyby`	Produce a grouped quantile column, as in `data.table`. `keyby` will set a key on the result (*i.e.* order by `keyby`).
`new.col`	If not `NULL`, the name of the column to be added. If `NULL` (the default) a name will be inferred from `n`. (For example, `n = 100` will be `<col>Percentile`).
`character.only`	(logical, default: `FALSE`) Do not contemplate `col` to be an unquoted column name.
`overwrite`	(logical, default: `TRUE`) If `TRUE` and `new.col` already exists in `DT`, the column will be overwritten. If `FALSE`, attempting to overwrite an existing column is an error.
`check.na`	(logical, default: `FALSE`) If `TRUE`, `NA`s in `DT[[col]]` will throw an error. If `NA`'s are present, the corresponding n-tile may take any value.

Value

`DT` with a new integer column `new.col` containing the quantiles. If `DT` is not a `data.table` its class may be preserved unless `keyby` is used, where it will always be a `data.table`.

Examples

```
library(data.table)
DT <- data.table(x = 1:20, y = 2:1)
mutate_ntile(DT, "x", n = 10)
mutate_ntile(DT, "x", n = 5)
mutate_ntile(DT, "x", n = 10, by = "y")
mutate_ntile(DT, "x", n = 10, keyby = "y")

y <- "x"
DT <- data.table(x = 1:20, y = 2:1)
mutate_ntile(DT, y, n = 5)                        # Use DT$y
mutate_ntile(DT, y, n = 5, character.only = TRUE) # Use DT$x
```

`mutate_other` *Group infrequent entries into 'Other category'*

Description

Useful when you want to constrain the number of unique values in a column by keeping only the most common values.

Usage

```
mutate_other(
  .data,
  var,
  n = 5,
  count,
  by = NULL,
  var.weight = NULL,
  mass = NULL,
  copy = TRUE,
  other.category = "Other"
)
```

Arguments

`.data`	Data containing variable.
`var`	Variable containing infrequent entries, to be collapsed into "Other".
`n`	Threshold for total number of categories above "Other".
`count`	Threshold for total count of observations before "Other".
`by`	Extra variables to group by when calculating `n` or `count`.
`var.weight`	Variable to act as a weight: `var`'s where the sum of this variable exceeds `mass` will be kept, others set to `other.category`.

`mass`	Threshold for sum of `var.weight`: any `var` where the aggregated sum of `var.weight` exceeds `mass` will be kept and other `var` will be set to `other.category`. By default (`mass = NULL`), the value of `mass` is $-\infty$, with a warning. You may set it explicitly to `-Inf` if you really want to avoid a warning that this function will have no effect.
`copy`	Should `.data` be copied? Currently only `TRUE` is supported.
`other.category`	Value that infrequent entries are to be collapsed into. Defaults to `"Other"`.

Value

`.data` but with `var` changed so that infrequent values have the same value (`other.category`).

Examples

```
library(data.table)
library(magrittr)

DT <- data.table(City = c("A", "A", "B", "B", "C", "D"),
                 value = c(1, 9, 4, 4, 5, 11))

DT %>%
  mutate_other("City", var.weight = "value", mass = 10) %>%
  .[]
```

`ngrep` *Anti-grep*

Description

It is not simple to negate a regular expression. This obviates the need takes the long way round: negating the corresponding `grepl` call.

Usage

```
ngrep(pattern, x, value = FALSE, ...)
```

Arguments

`x, value, pattern`	As in `grep`.
`...`	Arguments passed to `grepl`.

Value

If `value` is `FALSE` (the default), indices of `x` which do not match the pattern; if `TRUE`, the values of `x` themselves.

Examples

```
 grep("[a-h]", letters)
ngrep("[a-h]", letters)

txt <- c("The", "licenses", "for", "most", "software", "are",
"designed", "to", "take", "away", "your", "freedom",
"to", "share", "and", "change", "it.",
"", "By", "contrast,", "the", "GNU", "General", "Public", "License",
"is", "intended", "to", "guarantee", "your", "freedom", "to",
"share", "and", "change", "free", "software", "--",
"to", "make", "sure", "the", "software", "is",
"free", "for", "all", "its", "users")

 grep("[gu]", txt, value = TRUE)
ngrep("[gu]", txt, value = TRUE)
```

`prohibit_unequal_length_vectors` *Prohibit unequal length vectors*

Description

Tests whether all vectors have the same length.

Usage

```
prohibit_unequal_length_vectors(...)
```

Arguments

`...`	Vectors to test.

Value

An error message unless all of `...` have the same length in which case `NULL`, invisibly.

`prohibit_vector_recycling` *Prohibit vector recycling*

Description

Tests (harshly) whether the vectors can be recycled safely.

Usage

```
prohibit_vector_recycling(...)

prohibit_vector_recycling.MAXLENGTH(...)
```

Arguments

`...`	A list of vectors

Value

An error message if the vectors are of different length (unless the alternative length is 1). The functions differ in their return values on success: `prohibit_vector_recycling.MAXLENGTH` returns the maximum of the lengths whereas `prohibit_vector_recyling` returns `NULL`. (Both functions return their values invisibly.)

Examples

```
## Not run:
# Returns nothing because they are of the same length
prohibit_vector_recycling(c(2, 2), c(2, 2))
# Returns nothing also, because the only different length is 1
prohibit_vector_recycling(c(2, 2), 1)
# Returns an error:
prohibit_vector_recycling(c(2, 2), 1, c(3, 3, 3))

## End(Not run)
```

`provide.dir` *Provide directory*

Description

Provide directory. Create directory only if it does not exist.

Usage

```
provide.dir(path, ...)
```

Arguments

`path`	Path to create.
`...`	Passed to `dir.create`.

Value

`path` on success, the empty string `character(1)` on failure.

`provide.file` *Provide a file*

Description

Present since `hutils` `v1.5.0`.

Usage

```
provide.file(path, on_failure = "")
```

Arguments

`path`	A string. The path to a filename that requires existence.
`on_failure`	The return value on failure. By default, an empty string.

Value

`path` for success. Or `on_failure` if the `path` cannot be provided.

`replace_pattern_in` *Replace string pattern in text file*

Description

Replace string pattern in text file

Usage

```
replace_pattern_in(
  file_contents,
  replace,
  basedir = ".",
  dir_recursive = TRUE,
  reader = readLines,
  file_pattern = "\\.(R|r)(nw|md)?$",
  file_contents_perl = TRUE,
  file_contents_fixed = FALSE,
  file_contents_ignore_case = FALSE,
  writer = writeLines
)
```

Arguments

file_contents	Character string containing a regular expression to be matched in the given character vector. Passed to pattern in gsub.
replace	The replacement, passed to replacement in gsub.
basedir	The root of the directory tree in which files will be searched recursively.
dir_recursive	(logical, default: TRUE) Search within subdirectories of basedir?
reader	A function, akin to base::readLines, the default, that accepts a filename and returns a character vector.
file_pattern	A regular expression passed to list.files(pattern = file.ext). By default, "\.(R|r)(nw|md)?$", i.e. all R and Sweave files. (Does not have to be a file extension.)
file_contents_perl	(logical, default: TRUE) Should file_contents be interpreted as a perl regex?
file_contents_fixed	(logical, default: FALSE) Should file_contents be interpreted as a fixed regex?
file_contents_ignore_case	(logical, default: FALSE) As in grep.
writer	A function that will rewrite the file from the character vector read in.

---

report_error *Report errors and warnings*

Description

Provides a consistent style for errors and warnings.

Usage

```
report_error(
  faulty_input,
  error_condition,
  requirement,
  context = NULL,
  advice,
  hint = NULL,
  halt = TRUE
)
```

Arguments

`faulty_input`	Unquoted function argument that is the cause of the error condition.
`error_condition`	A sentence explaining the condition that invoked the error.
`requirement`	A sentence that explains what is required.
`context`	(Optional) A sentence that contextualizes the error
`advice`	Advice for the user to avoid the error.
`hint`	If the input can be guessed,
`halt`	(logical, default: `TRUE`) Should the function signal an error and halt?

`RQ` *Shorthand for* `requireNamespace`

Description

Present since `hutils` `v1.2.0`. Alias for `if (!requireNamespace(pkg, quietly = TRUE))` *yes* `else` *no*. Typical use-case would be `RQ(pkg, install.packages("pkg"))]`.

Default values for `yes` and `no` from `hutils` `v1.5.0`.

This function is not recommended for use in scripts as it is a bit cryptic; its use-case is for bash scripts and the like where calls like this would otherwise be frequent and cloud the message.

Usage

```
RQ(pkg, yes = NULL, no = NULL)
```

Arguments

`pkg`	Package to test whether the package is not yet installed.
`yes`	Response if `pkg` is **not** installed.
`no`	(optional) Response if `pkg` is installed.

Examples

```
## Not run:
 RQ("dplyr", "dplyr needs installing")

## End(Not run)
```

samp *Safer sampler*

Description

Present since hutils v1.4.0. Same as sample, but avoiding the behaviour when length(x) == 1L.

Usage

```
samp(x, size = length(x), replace = size > length(x), loud = TRUE, prob = NULL)
```

Arguments

x	A vector.
size	A non-negative integer, the number of items to return.
replace	Should the sampling be done with replacement? Defaults to TRUE if size > length(x), with a message.
loud	If TRUE, the default, any behaviour known to be different from sample is flagged with a message.
prob	As in sample.

Examples

```
samp(1:5)
sample(1:5)

samp(1:5, size = 10)  # no error
tryCatch(sample(1:5, size = 10),
         error = function(e) print(e$m))

samp(5, size = 3)
sample(5, size = 3)
```

selector *Fast selection of* data.table *columns*

Description

Present since hutils 1.2.0.

Usage

```
selector(DT, ..., cols = NULL, preserve.key = TRUE, shallow = FALSE)
```

Arguments

`DT`	A `data.table`.
`...`	Unquoted columns names.
`cols`	Character vector of column names.
`preserve.key`	(logical, default: `TRUE`) Reapply the key (if `DT` has one)?
`shallow`	(logical, default: `FALSE`) Should the result be a shallow `copy` of `DT`'s columns or should the columns be assigned by reference? If `TRUE`, any modification to the result also modifies the selected columns in `DT`.

Value

`DT` with the selected columns.

Examples

```
RQ("nycflights13", no = {
 library(nycflights13)
 library(data.table)
 fs <- as.data.table(flights)
 fs1 <- selector(fs, year, month, day, arr_delay)
 fs1[, arr_delay := NA]
})
```

`select_grep` *Select names matching a pattern*

Description

Select names matching a pattern

Usage

```
select_grep(
  DT,
  patterns,
  .and = NULL,
  .but.not = NULL,
  ignore.case = FALSE,
  perl = TRUE,
  fixed = FALSE,
  useBytes = FALSE,
  invert = FALSE,
  .warn.fixed.mismatch = TRUE
)
```

Arguments

`DT`	A `data.frame`.
`patterns`	Regular expressions to be matched against the names of `DT`. If `length(patterns) > 1` the patterns are concatenated using alternation.
`.and`	Character or integer positions of names to select, regardless of whether or not they are matched by `patterns`.
`.but.not`	Character or integer positions of names to drop, regardless of whether or not they are matched by `patterns` or whether they are explicitly added by `.and`.
`ignore.case, perl, fixed, useBytes, invert`	Arguments passed to `grep`. Note that `perl = TRUE` by default (unlike `grep`) unless `fixed = TRUE` (and `perl` is missing).
`.warn.fixed.mismatch`	(logical, default: `TRUE`) If `TRUE`, the default, selecting `fixed = TRUE` with `perl = TRUE` or `ignore.case = TRUE` results in `perl` and `ignore.case` being reset to `FALSE` with a warning (as in `grep`), even if it makes no difference to the columns eventually selected. If `FALSE` unambiguous results are allowed; if `ignore.case = TRUE` and `fixed = TRUE`, the result is **unambiguous** if `select_grep(DT, tolower(patterns), fixed = TRUE)` and `select_grep(DT, toupper(patterns), fixed = TRUE)` are identical.

Value

`DT` with the selected names.

integer vector of positions

Examples

```
library(data.table)
dt <- data.table(x1 = 1, x2 = 2, y = 0)
select_grep(dt, "x")
select_grep(dt, "x", .and = "y")
select_grep(dt, "x", .and = "y", .but.not = "x2")
```

`select_which` *Select columns satisfying a condition*

Description

Select columns satisfying a condition

Usage

```
select_which(DT, Which, .and.dots = NULL, checkDT = TRUE, .and.grep = NULL)
```

Arguments

`DT`	A `data.table`.
`Which`	A function that takes a vector and returns `TRUE` or `FALSE`. `TRUE` columns are selected.
`.and.dots`	Optional extra columns to include. May be a character vector of `names(DT)` or numeric (positions) or logical. If provided, the columns so added (if they do not satisfy `Which`) will be after all the columns `Which` do so satisfy.
`checkDT`	If `TRUE` (the default), an informative error message is provided if `DT` is not a `data.table`.
`.and.grep`	A character vector of regular expressions to match to the names of `DT`. The corresponding columns will be included in the result.

Value

`DT` with the selected variables.

Examples

```
library(data.table)
DT <- data.table(x = 1:5,
                 y = letters[1:5],
                 AB = c(NA, TRUE, FALSE))
select_which(DT, anyNA, .and.dots = "y")
```

`seq_nrow` *Generate sequence of row numbers*

Description

Generate sequence of row numbers

Usage

```
seq_nrow(x)
```

Arguments

`x`	An object that admits an `nrow`.

Value

Equivalent to `seq_len(nrow(x))`

set_cols_first *Put columns first or last*

Description

Reorder columns of a `data.table` (via `setcolorder`) so that particular columns appear first (or last), or in a particular order.

Usage

```
set_cols_first(DT, cols, intersection = TRUE)

set_cols_last(DT, cols, intersection = TRUE)

set_colsuborder(DT, cols, intersection = TRUE)
```

Arguments

`DT`	A data.table.
`cols`	Character vector of columns to put before (after) all others or, in the case of `set_colsuborder`, a vector of columns in the order requested.
`intersection`	Use the intersection of the names of `DT` and `cols`. If `FALSE` any `cols` are not the names of `DT`, the function may error on behalf of `data.table`. Not available for `set_colsuborder`.

Details

In the case of `set_colsuborder` the group of columns `cols` occupy the same positions in `DT` but in a different order. See examples.

Examples

```
library(data.table)

DT <- data.table(y = 1:5, z = 11:15, x = letters[1:5])
set_cols_first(DT, "x")[]
set_cols_last(DT, "x")[]
set_colsuborder(DT, c("x", "y"))[]
```

`swap` *Swap assignment*

Description

Swap values simultaneously. Present since `hutils 1.4.0`.

Usage

```
x %<->% value
```

Arguments

`x, value`	Objects whose values are to be reassigned by swapping.

Value

`NULL` invisibly. Called for its side-effect: the values of `x` and `value` are swapped. So

```
x %<->% value
```

is equivalent to

```
temp <- x
x <- value
value <- temp
rm(temp)
```

Examples

```
a <- 1
b <- 2
a %<->% b
a
b
```

`Switch` *Vectorized switch*

Description

Present since `hutils` `1.2.0`. Vectorized version of `switch`. Used to avoid or make clearer the result of `if_else(Expr == , ..1, if_else(Expr == , ..2, ...))`

Usage

```
Switch(Expr, ..., DEFAULT, IF_NA = NULL, MUST_MATCH = FALSE)
```

Arguments

`Expr`	A character vector.
`...`	As in `switch`, a list of named alternatives. Unlike `switch`, unnamed vectors are taken to match `""`. Likewise, `NA` values in `Expr` must be assigned via `IF_NA`.
`DEFAULT`	A mandatory default value should any name of `...` be left unmatched.
`IF_NA`	Optional value to replace missing (`NA_character_`) values in `Expr`.
`MUST_MATCH`	(logical, default: `FALSE`) Must every value in `Expr` be matched by a conversion in `...`? If `TRUE` any output equal to the value of `DEFAULT` is an error.

Value

For every element of `...` whose name matches an element of `Expr`, that element's value.

Examples

```
Switch(c("a", "b", "c", "a"),
       "a" = 1,
       "b" = 2,
       "c" = 3,
       "4" = 4,
       DEFAULT = 0)
```

`unique-keys` *Unique keys*

Description

A `data.table`'s `key` need not be unique, but there are frequently circumstances where non-unique keys can wreak havoc. `has_unique_key` reports the existence of a unique key, and `set_unique_key` both sets and ensures the uniqueness of keys.

Usage

```
has_unique_key(DT)

set_unique_key(DT, ...)
```

Arguments

`DT`	A data.table
`...`	keys to set

Value

`has_unique_key` returns `TRUE` if `DT` has a unique key, `FALSE` otherwise. `set_unique_key` runs `setkey(DT, ...)` then checks whether the key is unique, returning the keyed `data.table` if the key is unique, or an error message otherwise.

`weight2rows` *Expand a weighted data frame to an equivalent unweighted*

Description

Present since `v1.0.0`. Argument `rows.out` available since `v1.3.0`; `rows.out` < `1` supported since `v 1.4.0`. Argument `discard_weight.var` available since `v1.3.0`.

Usage

```
weight2rows(DT, weight.var, rows.out = NULL, discard_weight.var = FALSE)
```

Arguments

`DT`	A `data.table`. Will be converted to one if possible.
`weight.var`	Variable in `DT` to be used as weights.
`rows.out`	If not `NULL` (the default) specifies the number of rows in the result; otherwise the number of rows will be `sum(DT[[weight.var]])`. (Due to rounding, this figures are inexact.) Since `v1.4.0`, if `0` < `rows.out` < `1` then taken to be a sample of the unweighted table. (So `rows.out` = `0.1` would give a 10% sample.)
`discard_weight.var`	If `FALSE`, the default, `weight.var` in `DT` will be `1` for each row in the result or a new weight if `rows.out` is given. Otherwise, `TRUE` drops the column entirely.

Value

`DT` but with the number of rows expanded to `sum(DT[[weight.var]])` to reflect the weighting.

Examples

```
library(data.table)
DT <- data.table(x = 1:5, y = c(1, 1, 1, 1, 2))
weight2rows(DT, "y")
weight2rows(DT, "y", rows.out = 5)
```

`weighted_ntile` *Weighted (ranked) quantiles*

Description

Weighted (ranked) quantiles

Usage

```
weighted_ntile(vector, weights = rep(1, times = length(vector)), n)
```

Arguments

`vector`	The vector for which quantiles are desired.
`weights`	The weights associated with the vector. None should be `NA` or zero.
`n`	The number of quantiles desired.

Details

With a short-length vector, or with weights of a high variance, the results may be unexpected.

Value

A vector of integers corresponding to the ntiles. (As in `dplyr::ntile`.)

Examples

```
weighted_ntile(1:10, n = 5)
weighted_ntile(1:10, weights = c(rep(4, 5), rep(1, 5)), n = 5)
```

`weighted_quantile` *Weighted quantile*

Description

`quantile` when the values are weighted

Usage

```
weighted_quantile(v, w = NULL, p = (0:4)/4, v_is_sorted = FALSE)
```

Arguments

`v`	A vector from which sample quantiles are desired.
`w`	Weights corresponding to each `v`.
`p`	Numeric vector of probabilities. Missing values or values outside $[0, 1]$ raise an error.
`v_is_sorted`	(logical, default: `FALSE`) If `TRUE`, ordering `v` is assumed to be sorted. Only set to `TRUE` when it is certain that `v` is sorted (as within groups of tables).

Value

A vector the same length as `p`, the quantiles corresponding to each element of `p`.

`%ein%` *Exists and (not) in*

Description

A common blunder in R programming is to mistype one of a set of filters without realizing. This function will error if any member of the values to be matched against is not present.

Usage

```
lhs %ein% rhs

lhs %enotin% rhs
```

Arguments

`lhs`	Values to be matched
`rhs`	Values to be matched against.

Value

Same as `%in%` and `%notin%`, unless an element of `rhs` is not present in `lhs`, in which case, an error.

Examples

```
# Incorrectly assumed to include two Species
iris[iris$Species %in% c("setosa", "versicolour"), ]
## Not run:
# Error:
iris[iris$Species %ein% c("setosa", "versicolour"), ]

## End(Not run)
```

`%notchin%` *Negation of in (character)*

Description

Negation of in (character)

Usage

```
x %notchin% y
```

Arguments

`x`	Values to be matched.
`y`	Values to be matched against.

Details

If `y` is `NULL`, then `x` is `TRUE` for consistency with `%in%`. If `x` and `y` are not both character, the function simply falls back to `%in%` rather than erroring.

`%notin%` *Negation of in*

Description

Negation of in

Usage

```
x %notin% y
```

Arguments

`x`	Values to be matched
`y`	Values to be matched against.

Details

If `y` is `NULL`, then `x` is `TRUE` for consistency with `%in%`. Note that the function uses `fmatch` internally for performance on large `y`. Accordingly, `y` will be modified by adding a `.match.hash` attribute and thus must not be used in packages where `y` is a constant, or for things like names of `data.table`.

`%pin%` *Partial in*

Description

Analogue of `%in%` but indicating partial match of the left operand.

Usage

```
x %pin% Y
```

Arguments

`x`	The values to be matched. Same as `%in%`.
`Y`	A vector of values (perl regular expressions) to be matched against.

Value

`TRUE` for every `x` for which any `grepl` is `TRUE`.

Examples

```
x <- c("Sydney Airport", "Melbourne Airport")

x %pin% c("Syd", "Melb")
```

Index