Package 'iNZightTools'

October 12, 2023

Type Package

Title Tools for 'iNZight'

Version 2.0.1

Depends R (>= 4.0)

Imports dplyr (>= 1.1.0), DBI, forcats, glue, grDevices, magrittr, methods, purrr (>= 1.0.0), readr (>= 1.2.0), rlang (>= 0.4.9), srvyr, stats, stringr, survey, tibble, tidyr (>= 1.3.0), tools, units, utils

Suggests chron, covr, dbplyr, expss, haven, jsonlite, knitr, lubridate, RCurl, readxl, RSQLite, styler, surveyspec, testthat (>= 3.0.0), tsibble, validate, yaml

Additional_repositories https://r.docker.stat.auckland.ac.nz

BugReports https://github.com/iNZightVIT/iNZightTools/issues

Contact inzight_support@stat.auckland.ac.nz

URL https://inzight.nz

Description Provides a collection of wrapper functions for common variable and dataset manipulation workflows primarily used by 'iNZight', a graphical user interface providing easy exploration and visualisation of data for students of statistics, available in both desktop and online versions. Additionally, many of the functions return the 'tidyverse' code used to obtain the result in an effort to bridge the gap between GUI and coding.

License GPL-3

Encoding UTF-8

Language en-GB

RoxygenNote 7.2.3

Config/testthat/edition 3

NeedsCompilation no

Author Tom Elliott [aut, cre] (<https://orcid.org/0000-0002-7815-6318>),
Daniel Barnett [aut],
Yiwen He [aut],
Zhaoming Su [aut],

Lushi Cai [ctb],
Akshay Gupta [ctb],
Owen Jin [ctb],
Christoph Knopf [ctb]

Maintainer Tom Elliott <tom.elliott@auckland.ac.nz>

Repository CRAN

Date/Publication 2023-10-12 11:50:08 UTC

R topics documented:

`add_suffix` *Add suffix to string*

Description

When creating new variables or modifying the data set, we often add a suffix added to distinguish the new name from the original one. However, if the same action is performed twice (for example, filtering a data set), the suffix is duplicated (data.filtered.filtered). This function averts this by adding the suffix if it doesn't exist, and otherwise appending a counter (data.filtered2).

Usage

```
add_suffix(name, suffix)
```

Arguments

`name`	a character vector containing (original) names
`suffix`	the suffix to add, a length-one character vector

Value

character vector of names with suffix appended

Examples

```
add_suffix("data", "filtered")
add_suffix(c("data.filtered", "data.filtered.reshaped"), "filtered")
```

`aggregate_data` *Aggregate data by categorical variables*

Description

Summarizes non-categorical variables in a dataframe by grouping them based on specified categorical variables and returns the aggregated result along with the tidyverse code used to generate it.

Usage

```
aggregate_data(
  data,
  group_vars,
  summaries,
  vars = NULL,
  names = NULL,
  quantiles = c(0.25, 0.75)
)

aggregate_dt(
  data,
  dt,
  dt_comp,
  group_vars = NULL,
  summaries,
  vars = NULL,
  names = NULL,
  quantiles = c(0.25, 0.75)
)
```

Arguments

`data`	A dataframe or survey design object to be aggregated.
`group_vars`	A character vector specifying the variables in `data` to ' be used as grouping factors.

summaries	An unnamed character vector or named list of summary functions to calculate for each group. If unnamed, the vector elements should be names of variables in the dataset for which summary statistics need to be calculated. If named, the names should correspond to the summary functions (e.g., "mean", "sd", "iqr") to be applied to each variable.
vars	(Optional) A character vector specifying the names of variables in the dataset for which summary statistics need to be calculated. This argument is ignored if summaries is a named list.
names	(Optional) A character vector or named list providing name templates for the newly created variables. See details for more information.
quantiles	(Optional) A numeric vector specifying the desired quantiles (e.g., c(0.25, 0.5, 0.75)). See details for more information.
dt	A character string representing the name of the date-time variable in the dataset.
dt_comp	A character string specifying the component of the date-time to use for grouping.

Details

The aggregate_data() function accepts any R function that returns a single-value summary (e.g., mean, var, sd, sum, IQR). By default, new variables are named {var}_{fun}, where {var} is the variable name and {fun} is the summary function used. The user can provide custom names using the names argument, either as a vector of the same length as vars, or as a named list where the names correspond to summary functions (e.g., "mean" or "sd").

The special summary "missing" can be included, which counts the number of missing values in the variable. The default name for this summary is {var}_missing.

If quantiles are requested, the function calculates the specified quantiles (e.g., 25th, 50th, 75th percentiles), creating new variables for each quantile. To customize the names of these variables, use {p} as a placeholder in the names argument, where {p} represents the quantile value. For example, using names = "Q{p}_{var}" will create variables like "Q0.25_Sepal.Length" for the 25th percentile.

Value

An aggregated dataframe containing the summary statistics for each group, along with the tidyverse code used for the aggregation.

Functions

- aggregate_dt(): Aggregate data by dates and times

Author(s)

Tom Elliott, Owen Jin, Zhaoming Su

Zhaoming Su

See Also

code

aggregate_data

Examples

```
aggregated <-
    aggregate_data(iris,
        group_vars = c("Species"),
        summaries = c("mean", "sd", "iqr")
    )
code(aggregated)
head(aggregated)
```

---

append_rows *Append rows to a dataset*

Description

Append rows to a dataset

Usage

```
append_rows(data, new_data, when_added = FALSE)
```

Arguments

data	The original dataset to which new rows will be appended.
new_data	The dataset containing the new rows.
when_added	Logical; indicates whether a .when_added column is required.

Value

A dataset with new rows appended below the original data.

Author(s)

Yiwen He, Zhaoming Su

code *Get Data's Code*

Description

Used to grab code from a data.frame generated by this package.

Usage

```
code(data)
```

Arguments

data	dataset you want to extract the code from

Details

This is simply a helper function to grab the contents of the 'code' attribute contained in the data object.

Value

The code used to generate the data.frame, if available (else NULL)

Author(s)

Tom Elliott

collapse_cat *Collapse data by values of a categorical variable*

Description

Collapse values in a categorical variable into one defined level

Usage

```
collapse_cat(data, var, levels, new_level, name = NULL)
```

Arguments

data	a dataframe to collapse
var	a string of the name of the categorical variable to collapse
levels	a character vector of the levels to be collapsed
new_level	a string for the new level
name	a name for the new variable

Value

the original dataframe containing a new column of the collapsed variable with tidyverse code attached

Author(s)

Zhaoming Su

See Also

code

Examples

```
collapsed <- collapse_cat(iris,
    var = "Species",
    c("versicolor", "virginica"),
    new_level = "V"
)
cat(code(collapsed))
tail(collapsed)
```

---

combine_vars *Combine variables into one categorical variable*

Description

Combine chosen variables of any class by concatenating them into one factor variable, and returns the result along with tidyverse code used to generate it.

Usage

```
combine_vars(
  data,
  vars,
  sep = ":",
  name = NULL,
  keep_empty = FALSE,
  keep_na = TRUE
)
```

Arguments

data	a dataframe with the columns to be combined
vars	a character vector of the variables to be combined
sep	a character string to separate the levels

name	a name for the new variable
keep_empty	logical, if FALSE empty level combinations are removed from the factor
keep_na	logical, if TRUE the <NA> in the factors or NA in the characters will turn in a level "(Missing)"; otherwise, the resulting entries will return <NA>

Value

original dataframe containing new columns of the new categorical variable with tidyverse code attached

Author(s)

Owen Jin, Zhaoming Su

Examples

```
combined <- combine_vars(warpbreaks, vars = c("wool", "tension"), sep = "_")
cat(code(combined))
head(combined)
```

---

convert_to_cat *Convert variables to categorical variables*

Description

Convert specified variables into factors

Usage

```
convert_to_cat(data, vars, names = NULL)
```

Arguments

data	a dataframe with the categorical column to convert
vars	a character vector of column names to convert
names	a character vector of names for the created variables

Value

original dataframe containing new columns of the converted variables with tidyverse code attached

Author(s)

Zhaoming Su

See Also

code

Examples

```
converted <- convert_to_cat(iris, vars = c("Petal.Width"))
cat(code(converted))
head(converted)
```

---

convert_to_date *Convert variables to dates*

Description

Convert variables to dates

Usage

```
convert_to_date(data, vars, ord = NULL, names = NULL)
```

Arguments

data	a dataframe with the variables to convert
vars	a character vector of column names to convert
ord	a character vector of date-time formats
names	a character vector of names for the created variables

Value

original dataframe containing new columns of the converted variables with tidyverse code attached

Author(s)

Zhaoming Su

See Also

code

convert_to_datetime *Convert variables to date-time*

Description

Convert variables to date-time

Usage

```
convert_to_datetime(data, vars, ord = NULL, names = NULL, tz = "")
```

Arguments

data	a dataframe with the variables to convert
vars	a character vector of column names to convert
ord	a character vector of date-time formats
names	a character vector of names for the created variables
tz	a time zone name (default: local time zone). See OlsonNames

Value

original dataframe containing new columns of the converted variables with tidyverse code attached

Author(s)

Zhaoming Su

See Also

code

create_varname *Create variable name*

Description

Convert a given string to a valid R variable name, converting spaces to underscores (_) instead of dots.

Usage

```
create_varname(x)
```

Arguments

x	a string to convert

Value

a string, which is also a valid variable name

Author(s)

Tom Elliott

Examples

```
create_varname("a new variable")
create_varname("8d4-2q5")
```

`create_vars` *Create new variables*

Description

Create new variables by using valid R expressions and returns the result along with tidyverse code used to generate it.

Usage

```
create_vars(data, vars = ".new_var", vars_expr = NULL)
```

Arguments

`data`	a dataframe to which to add new variables to
`vars`	a character of the new variable names
`vars_expr`	a character of valid R expressions which can generate vectors of values

Value

original dataframe containing the new columns created from `vars_expr` with tidyverse code attached

Author(s)

Zhaoming Su

See Also

`code`

Examples

```
created <- create_vars(
    data = iris,
    vars = "Sepal.Length_less_Sepal.Width",
    "Sepal.Length - Sepal.Width"
)
cat(code(created))
head(created)
```

`delete_vars` *Delete variables*

Description

Delete variables from a dataset

Usage

```
delete_vars(data, vars = NULL)
```

Arguments

`data`	dataset
`vars`	variable names to delete

Value

dataset without chosen variables

Author(s)

Zhaoming Su

`extract_dt_comp` *Extract date component from a date-time variable*

Description

This function extracts a specific date component from a date-time variable in a dataframe.

Usage

```
extract_dt_comp(data, var, comp, name = NULL)
```

Arguments

data	The dataframe containing the date-time variable.
var	The name of the date-time variable to extract the component.
comp	The date component wanted from the variable. See iNZightTools:::inz_dt_comp for the full list of components.
name	The name of the new column to store the extracted date component.

Value

A dataframe with the new date component column.

Author(s)

Zhaoming Su

---

extract_part *Extract part of a datetimes variable (DEPRECATED)*

Description

This function has been replaced by ‘extract_dt_comp‘ and will be removed in the next release.

Usage

```
extract_part(.data, varname, part, name)
```

Arguments

.data	dataframe
varname	name of the variable
part	part of the variable wanted
name	name of the new column

Value

see ‘extract_dt_comp‘

filter *Filter*

Description

Filter

Filter inzdf

Usage

```
## S3 method for class 'inzdf_db'
filter(.data, ..., table = NULL, .preserve = FALSE)
```

Arguments

.data	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr). See *Methods*, below, for more details.
...	<data-masking> Expressions that return a logical value, and are defined in terms of the variables in .data. If multiple expressions are included, they are combined with the & operator. Only rows for which all conditions evaluate to TRUE are kept.
table	name of the table to use, defaults to first in list
.preserve	ignored

filter_cat *Filter data by levels of categorical variables*

Description

This function filters a dataframe or survey design object by keeping only the rows where a specified categorical variable matches one of the given levels. The resulting filtered dataframe is returned, along with the tidyverse code used to generate it.

Usage

```
filter_cat(data, var, levels)
```

Arguments

data	A dataframe or survey design object to be filtered.
var	The name of the column in data to be filtered by.
levels	A character vector of levels in var to keep.

Value

A filtered dataframe with the tidyverse code attached.

Author(s)

Owen Jin, Zhaoming Su

See Also

code

Examples

```
filtered <- filter_cat(iris,
    var = "Species",
    levels = c("versicolor", "virginica")
)
cat(code(filtered))
head(filtered)
```

`filter_num` *Filter data by levels of numeric variables*

Description

This function filters a dataframe or survey design object by applying a specified boolean condition to one of its numeric variables. The resulting filtered dataframe is returned, along with the tidyverse code used to generate it.

Usage

```
filter_num(data, var, op = c("<=", "<", ">=", ">", "==", "!="), num)
```

Arguments

`data`	A dataframe or survey design object to be filtered.
`var`	The name of the column in `data` to be filtered by.
`op`	A logical operator to apply for the filtering condition. Valid options are: "<=", "<", ">=", ">", "==", or "!=".
`num`	The numeric value for which the specified `op` is applied.

Value

A filtered dataframe with the tidyverse code attached.

Author(s)

Owen Jin, Tom Elliott, Zhaoming Su

See Also

code

Examples

```
filtered <- filter_num(iris, var = "Sepal.Length", op = "<=", num = 5)
cat(code(filtered))
head(filtered)

library(survey)
data(api)
svy <- svydesign(~ dnum + snum,
    weights = ~pw, fpc = ~ fpc1 + fpc2,
    data = apiclus2
)
svy_filtered <- filter_num(svy, var = "api00", op = "<", num = 700)
cat(code(svy_filtered))
```

---

fitDesign *Fit a survey design*

Description

Fit a survey design to an object

Usage

```
fitDesign(svydes, dataset.name)
```

Arguments

svydes	a design
dataset.name	a dataset name

Value

a survey object

Author(s)

Tom Elliott

`fitModel` *Fit models*

Description

Wrapper function for ‘lm‘, ‘glm‘, and ‘svyglm‘.

Usage

```
fitModel(
  y,
  x,
  data,
  family = "gaussian",
 link = switch(family, gaussian = "gaussian", binomial = "logit", poisson = "log",
    negbin = "log"),
  design = "simple",
  svydes = NA,
  surv_params = NULL,
  ...
)
```

Arguments

`y`	character string representing the response,
`x`	character string of the explanatory variables,
`data`	name of the object containing the data.
`family`	gaussian, binomial, poisson (so far, no others will be added)
`link`	the link function to use
`design`	data design specification. one of ’simple’, ’survey’ or ’experiment’
`svydes`	a vector of arguments to be passed to the svydesign function, excluding data (defined above)
`surv_params`	a vector containing arguments for `survival::Surv()`
`...`	further arguments to be passed to lm, glm, svyglm, such as offset, etc.

Value

A model call formula (using lm, glm, or svyglm)

Author(s)

Tom Elliott

`form_class_intervals` *Form Class Intervals*

Description

This function creates categorical intervals from a numeric variable in the given dataset.

Usage

```
form_class_intervals(
  data,
  variable,
  method = c("equal", "width", "count", "manual"),
  n_intervals = 4L,
  interval_width,
  format = "(a,b]",
  range = NULL,
  format_lowest = ifelse(isinteger, "< a", "<= a"),
  format_highest = "> b",
  break_points = NULL,
  name = sprintf("%s.f", variable)
)
```

Arguments

`data`	A dataset or a survey object.
`variable`	The name of the numeric variable to convert into intervals.
`method`	The method used to create intervals: • 'equal' for equal-width intervals, • 'width' for intervals of a specific width, • 'count' for equal-count intervals, and • 'manual' to specify break points manually.
`n_intervals`	For methods 'equal' and 'count', this specifies the number of intervals to create.
`interval_width`	For method 'width', this sets the width of the intervals.
`format`	The format for interval labels; use 'a' and 'b' to represent the min/max of each interval, respectively.
`range`	The range of the data; use this to adjust the labels (e.g., for continuous data, set this to the floor/ceiling of the min/max of the data to get prettier intervals). If range does not cover the range of the data, values outside will be placed into 'less than a' and 'greater than b' categories.
`format_lowest`	Label format for values lower than the min of range.
`format_highest`	Label format for values higher than the max of range.
`break_points`	For method 'manual', specify breakpoints here as a numeric vector.
`name`	The name of the new variable in the resulting data set.

Value

A dataframe with an additional column containing categorical class intervals.

Author(s)

Tom Elliott, Zhaoming Su

Examples

```
form_class_intervals(iris, "Sepal.Length", "equal", 5L)
```

`inzdf` *iNZight data frame object*

Description

This object allows the data to be either a standard R `data.frame` or a connection to a database.

Usage

```
inzdf(x, name, ...)

## S3 method for class 'tbl_df'
inzdf(x, name, ...)

## S3 method for class 'data.frame'
inzdf(x, name, ...)

## S3 method for class 'SQLiteConnection'
inzdf(
  x,
  name = deparse(substitute(x)),
  schema = NULL,
  var_attrs = list(),
  dictionary = NULL,
  keep_con = FALSE,
  ...
)
```

Arguments

`x`	a data.frame or db connection
`name`	the name of the data
`...`	additional arguments passed to methods
`schema`	a list specifying the schema of the database (used for linking)
`var_attrs`	nested list of variables attributes for each table > variable
`dictionary`	an inzdict object
`keep_con`	if ‘TRUE‘ data will remain in DB (use for very large data)

Details

TODO: It is possible to specify a linking structure between multiple datasets, and when variables are selected the dataset will be linked 'on-the-fly'. This, when used with databases, will significantly reduce the size of data in memory.

Value

an `inzdf` object

`is_cat` *Is factor check*

Description

This function checks if a variable a factor.

Usage

```
is_cat(x)
```

Arguments

`x`	the variable to check

Value

logical, `TRUE` if the variable is a factor

Author(s)

Tom Elliott

`is_dt` *Is datetime check*

Description

This function checks if a variable a date/time/datetime

Usage

```
is_dt(x)
```

Arguments

`x`	the variable to check

Value

logical, TRUE if the variable is a datetime

Author(s)

Tom Elliott

---

is_num *Is numeric check*

Description

This function checks if a variable is numeric, or could be considered one. For example, dates and times can be treated as numeric, so return TRUE.

Usage

```
is_num(x)
```

Arguments

x	the variable to check

Value

logical, TRUE if the variable is numeric

Author(s)

Tom Elliott

---

is_preview *Is Preview*

Description

Checks if the complete file was read or not.

Usage

```
is_preview(df)
```

Arguments

df	data to check

Value

logical

is_survey *Check if object is a survey object (either standard or replicate design)*

Description

Check if object is a survey object (either standard or replicate design)

Usage

```
is_survey(x)
```

Arguments

x	object to be tested

Value

logical

Author(s)

Tom Elliott

is_svydesign *Check if object is a survey object (created by svydesign())*

Description

Check if object is a survey object (created by svydesign())

Usage

```
is_svydesign(x)
```

Arguments

x	object to be tested

Value

logical

Author(s)

Tom Elliott

is_svyrep *Check if object is a replicate survey object (created by svrepdesign())*

Description

Check if object is a replicate survey object (created by svrepdesign())

Usage

```
is_svyrep(x)
```

Arguments

x	object to be tested

Value

logical

Author(s)

Tom Elliott

join_data *Join data with another dataset*

Description

Join data with another dataset

Usage

```
join_data(
  data_l,
  data_r,
  by = NULL,
  how = c("inner", "left", "right", "full", "anti", "semi"),
  suffix_l = ".x",
  suffix_r = ".y"
)
```

Arguments

data_l	original data
data_r	imported dataset
by	a character vector of variables to join by
how	the method used to join the datasets
suffix_l	suffix for the original dataset (ignored for filter-joins)
suffix_r	suffix for the imported dataset (ignored for filter-joins)

Value

joined dataset

Author(s)

Zhaoming Su

See Also

code, mutate-joins, filter-joins

---

load_linked *Import linked data into an* inzdf *object*

Description

Import linked data into an inzdf object

Usage

```
load_linked(
  x,
  schema,
  con,
  name = ifelse(missing(con), deparse(substitute(x)), deparse(substitute(con))),
  keep_con = FALSE,
  progress = FALSE,
  ...
)
```

Arguments

x	a linked specification file or vector of data set paths
schema	a list describing the schema/relationships between the files
con	a database connection to load the linked data into
name	the name of the data set collection
keep_con	if TRUE data will remain in DB (use for very large data)
progress	either TRUE or FALSE to enable/disable the default progress bar, or a list of three functions to x <- create(from, to), set(x, i), and destroy(x) a progress bar.
...	additional arguments passed to data reading function smart_read()

Value

an inzdf object

---

load_rda *Load object(s) from an Rdata file*

Description

Load object(s) from an Rdata file

Usage

```
load_rda(file)
```

Arguments

file	path to an rdata file

Value

list of data frames, plus code

Author(s)

Tom Elliott

See Also

save_rda

make_names *Make unique variable names*

Description

Helper function to create new variable names that are unique given a set of existing names (in a data set, for example). If a variable name already exists, a number will be appended.

Usage

```
make_names(new, existing = character())
```

Arguments

new	a vector of proposed new variable names
existing	a vector of existing variable names

Value

a vector of unique variable names

Author(s)

Tom Elliott

Examples

```
make_names(c("var_x", "var_y"), c("var_x", "var_z"))
```

missing_to_cat *Convert missing values to categorical variables*

Description

Turn <NA> in categorical variables into "(Missing)"; numeric variables will be converted to categorical variables where numeric values as "(Observed)" and NA as "(Missing)".

Usage

```
missing_to_cat(data, vars, names = NULL)
```

Arguments

data	a dataframe with the columns to convert its missing values into categorical
vars	a character vector of the variables in data for conversion of missing values
names	a character vector of names for the new variables

Value

original dataframe containing new columns of the converted variables for the missing values with tidyverse code attached

Author(s)

Zhaoming Su

See Also

code

Examples

```
missing <- missing_to_cat(iris, vars = c("Species", "Sepal.Length"))
cat(code(missing))
head(missing)
```

---

newdevice *Open a New Graphics Device*

Description

Opens a new graphics device

Usage

```
newdevice(width = 7, height = 7, ...)
```

Arguments

width	the width (in inches) of the new device
height	the height (in inches) of the new device
...	additional arguments passed to the new device function

Details

Depending on the system, difference devices are better. The windows device works fine (for now), only attempt to speed up any other devices that we're going to be using. We speed them up by getting rid of buffering.

Author(s)

Tom Elliott

`Not In operator` *Anti value matching*

Description

Anti value matching

Usage

```
x %notin% table
```

Arguments

`x`	vector of values to be matched
`table`	vector of values to match against

Value

A logical vector of same length as 'x', indicating if each element does **not** exist in the table.

`Or NULL operator` *NULL or operator*

Description

NULL or operator

Usage

```
a %||% b
```

Arguments

`a`	an object, potentially NULL
`b`	an object

Value

a if a is not NULL, otherwise b

print_code *Tidy-printing of the code attached to an object*

Description

Tidy-printing of the code attached to an object

Usage

```
print_code(x, ...)
```

Arguments

x	a dataframe with code attached
...	additional arguments passed to tidy_all_code()

Value

Called for side-effect of printing code to the console.

Examples

```
iris_agg <- aggregate_data(iris, group_vars = "Species", summaries = "mean")
print_code(iris_agg)
```

random_sample *Random sampling without replacement*

Description

Take a specified number of groups of observations with fixed group size by sampling without replacement and returns the result along with tidyverse code used to generate it.

Usage

```
random_sample(data, n, sample_size)
```

Arguments

data	a dataframe to sample from
n	the number of groups to generate
sample_size	the size of each group specified in n

Value

a dataframe containing the random samples with tidyverse code attached

Author(s)

Owen Jin, Zhaoming Su

See Also

code

Examples

```
rs <- random_sample(iris, n = 5, sample_size = 3)
cat(code(rs))
head(rs)
```

---

rank_vars *Rank the data of numeric variables*

Description

Rank the values of numeric variables, for example, in descending order, and then returns the result along with tidyverse code used to generate it. See row_number and percent_rank.

Usage

```
rank_vars(data, vars, rank_type = c("min", "dense", "percent"))
```

Arguments

data	a dataframe with the variables to rank
vars	a character vector of numeric variables in data to rank
rank_type	either "min", "dense" or "percent", see row_number, percent_rank

Value

the original dataframe containing new columns with the ranks of the variables in vars with tidyverse code attached

Author(s)

Zhaoming Su

See Also

code

Examples

```
ranked <- rank_vars(iris, vars = c("Sepal.Length", "Petal.Length"))
cat(code(ranked))
head(ranked)
```

`read_dictionary` *Data Dictionaries*

Description

This function reads a data dictionary from a file and attaches it to a dataset. The attached data dictionary provides utility functions that can be used by other methods, such as plots, to automatically create axes and more.

Usage

```
read_dictionary(
  file,
  name = "name",
  type = "type",
  title = "title",
  description = "description",
  units = "units",
  codes = "codes",
  values = "values",
  level_separator = "|",
  ...
)

## S3 method for class 'dictionary'
print(x, kable = FALSE, include_other = TRUE, ...)

## S3 method for class 'dictionary'
x[i, ...]

apply_dictionary(data, dict)

has_dictionary(data)

get_dictionary(data)
```

Arguments

`file`	The path to the file containing the data dictionary.
`name`	The name of the column containing the variable name.

type	The name of the column containing the variable type.
title	The name of the column containing a short, human-readable title for the variable. If blank, the variable name will be used instead.
description	The name of the column containing the variable description.
units	The name of the column containing units (for numeric variables only).
codes	The name of the column containing factor codes (for categorical variables only).
values	The name of the column containing factor values corresponding to the codes. These should be in the same order as the codes.
level_separator	The separator used to separate levels in codes and values columns. The default separator is "|". Alternatively, you can provide a vector of length 2, where the first element is used for codes and the second element for values.
...	Additional arguments, passed to smart_read.
x	A dictionary object.
kable	If TRUE, the output will be formatted using kable.
include_other	If TRUE, additional variables will be included in the output.
i	Subset index.
data	A dataset (dataframe, tibble).
dict	A dictionary (created using read_dictionary()).

Value

The dataset with the attached data dictionary.

`read_meta` *Read CSV with iNZight metadata*

Description

This function will read a CSV file with iNZight metadata in the header. This allows plain text CSV files to be supplied with additional comments that describe the structure of the data to make import and data handling easier.

Usage

```
read_meta(file, preview = FALSE, column_types, ...)
```

Arguments

file	the plain text file with metadata
preview	logical, if TRUE only the first 10 rows are returned
column_types	optional column types
...	more arguments

Details

The main example is to define factor levels for an integer variable in large data sets.

Value

a data frame

Author(s)

Tom Elliott

`read_text` *Read text as data*

Description

The text can also be the value ‘"clipboard"‘ which will use ‘readr::clipboard()‘.

Usage

```
read_text(txt, delim = "\t", ...)
```

Arguments

`txt`	character string
`delim`	the delimiter to use, passed to ‘readr::read_delim()‘
`...`	additional arguments passed to ‘readr::read_delim()‘

Value

data.frame

Author(s)

Tom Elliott

`remove_rows` *Remove rows from data by row numbers*

Description

This function filters a dataframe or a survey design object by removing specified rows based on the provided row numbers. The resulting filtered dataframe is returned, along with the tidyverse code used to generate it.

Usage

```
remove_rows(data, rows)
```

Arguments

`data`	A dataframe or a survey design object to be filtered.
`rows`	A numeric vector of row numbers to be sliced off.

Value

A filtered dataframe with the tidyverse code attached.

Author(s)

Owen Jin, Zhaoming Su

See Also

code

Examples

```
data <- remove_rows(iris, rows = c(1, 4, 5))
cat(code(data))
head(data)
```

`rename_levels` *Rename the levels of a categorical variable*

Description

Rename the levels of a categorical variables, and returns the result along with tidyverse code used to generate it.

Usage

```
rename_levels(data, var, tobe_asis, name = NULL)
```

Arguments

`data`	a dataframe with the column to be renamed
`var`	a character of the categorical variable to rename
`tobe_asis`	a named list of the old level names assigned to the new level names ie. list('new level names' = 'old level names')
`name`	a name for the new variable

Value

original dataframe containing a new column of the renamed categorical variable with tidyverse code attached

Author(s)

Zhaoming Su

See Also

code

Examples

```
renamed <- rename_levels(iris,
    var = "Species",
    tobe_asis = list(set = "setosa", ver = "versicolor")
)
cat(code(renamed))
head(renamed)
```

rename_vars *Rename column names*

Description

Rename columns of a dataset with desired names

Usage

```
rename_vars(data, tobe_asis)
```

Arguments

data	a dataframe with columns to rename
tobe_asis	a named list of the old column names assigned to the new column names ie. list('new column names' = 'old column names')

Value

original dataframe containing new columns of the renamed columns with tidyverse code attached

Author(s)

Zhaoming Su

See Also

code

Examples

```
renamed <- rename_vars(iris, list(
    sepal_length = "Sepal.Length",
    sepal_width = "Sepal.Width",
    petal_length = "Petal.Length",
    petal_width = "Petal.Width"
))
cat(code(renamed))
head(renamed)
```

`reorder_levels` *Reorder the levels of a categorical variable*

Description

Reorder the levels of a categorical variable either manually or automatically

Usage

```
reorder_levels(
  data,
  var,
  new_levels = NULL,
  auto = c("freq", "order", "seq"),
  name = NULL
)
```

Arguments

`data`	a dataframe to reorder
`var`	a categorical variable to reorder
`new_levels`	a character vector of the new factor order; overrides `auto` if not `NULL`
`auto`	only meaningful if `new_levels` is `NULL`: the method to auto-reorder the levels, see `fct_inorder`
`name`	name for the new variable

Value

original dataframe containing a new column of the reordered categorical variable with tidyverse code attached

Author(s)

Zhaoming Su

See Also

`code`

Examples

```
reordered <- reorder_levels(iris,
    var = "Species",
    new_levels = c("versicolor", "virginica", "setosa")
)
cat(code(reordered))
head(reordered)
```

```
reordered <- reorder_levels(iris,
    var = "Species",
    auto = "freq"
)
cat(code(reordered))
head(reordered)
```

---

reshape_data *Reshaping dataset from wide to long or from long to wide*

Description

Reshaping dataset from wide to long or from long to wide

Usage

```
reshape_data(
  data,
  data_to = c("long", "wide"),
  cols,
  names_to = "name",
  values_to = "value",
  names_from = "name",
  values_from = "value"
)
```

Arguments

data	a dataset to reshape
data_to	whether the target dataset is long or wide
cols	columns to gather together (for wide to long)
names_to	name for new column containing old names (for wide to long)
values_to	name for new column containing old values (for wide to long)
names_from	column to spread out (for long to wide)
values_from	values to be put in the spread columns (for long to wide)

Value

reshaped dataset

Author(s)

Zhaoming Su

save_rda *Save an object with, optionally, a (valid) name*

Description

Save an object with, optionally, a (valid) name

Usage

```
save_rda(data, file, name)
```

Arguments

data	the data frame to save
file	where to save it
name	optional, the name the data will have in the rda file

Value

logical, should be TRUE, along with code for the save

Author(s)

Tom Elliott

See Also

load_rda

select *Select*

Description

Select

select_vars *Select variables from a dataset*

Description

Select a (reordered) subset of variables from a subset.

Usage

```
select_vars(data, keep)
```

Arguments

data	the dataset
keep	vector of variable names to keep

Value

a data frame with tidyverse code attribute

Author(s)

Tom Elliott, Zhaoming Su

Examples

```
select_vars(iris, c("Sepal.Length", "Species", "Sepal.Width"))
```

separate_var *Separate columns*

Description

Separate columns

Usage

```
separate_var(data, var, by, names, into = c("cols", "rows"))
```

Arguments

data	dataset
var	name of variable to be separated
by	a string as delimiter between values (separate by delimiter) or integer(s) as number of characters to split by (separate by position), the length of by should be 1 unless by is integer and into = "cols"; if by is a non-integer numeric vector its values will be rounded down to the nearest integer
names	for into = "cols", a character vector of output column names; use NA if there are components that you don't want to appear in the output; the number of non-NA elements determines the number of new columns in the result
into	whether to split into new rows or columns

Value

Separated dataset

Author(s)

Zhaoming Su

---

sheets *List available sheets within a file*

Description

Useful when reading an Excel file to quickly check what other sheets are available.

Usage

```
sheets(x)
```

Arguments

x	a dataframe, presumably returned by smart_read

Value

vector of sheet names, or NULL if the file was not an Excel workbook

Author(s)

Tom Elliott

Examples

```
cas_file <- system.file("extdata/cas500.xls", package = "iNZightTools")
cas <- smart_read(cas_file)
sheets(cas)
```

`smart_read` *Read a data file*

Description

A simple function that imports a file without the users needing to specify information about the file type (see Details for more). The `smart_read()` function uses the file's extension to determine the appropriate function to read the data. Additionally, characters are converted to factors by default, mostly for compatibility with iNZight (https://inzight.nz).

Usage

```
smart_read(
  file,
  ext = tools::file_ext(file),
  preview = FALSE,
  column_types = NULL,
  ...
)
```

Arguments

`file`	the file path to read
`ext`	file extension, namely "csv" or "txt"
`preview`	logical, if `TRUE` only the first few rows of the data will be returned
`column_types`	vector of column types (see ?readr::read_csv)
`...`	additional parameters passed to read_* functions

Details

Currently, `smart_read()` understands the following file types:

- delimited (.csv, .txt)
- Excel (.xls, .xlsx)
- SPSS (.sav)
- Stata (.dta)
- SAS (.sas7bdat, .xpt)
- R data (.rds)
- JSON (.json)

Value

A dataframe with some additional attributes:

- `name` is the name of the file
- `code` contains the 'tidyverse' code used to read the data
- `sheets` contains names of sheets if 'file' is an Excel file (can be retrieved using the `sheets()` helper function)

Reading delimited files

By default, `smart_read()` will detect the delimiter used in the file if the argument `delimiter = NULL` is passed in (the default). If this does not work, you can override this argument:

```
smart_read('path/to/file', delimiter = '+')
```

Author(s)

Tom Elliott

`sort_vars` *Sort data by variables*

Description

Sorts a dataframe by one or more variables, and returns the result along with tidyverse code used to generate it.

Usage

```
sort_vars(data, vars, asc = rep(TRUE, length(vars)))
```

Arguments

`data`	a dataframe to sort
`vars`	a character vector of variable names to sort by
`asc`	logical, length of 1 or same length as `vars`. If `TRUE` (default), then sorted in ascending order, otherwise descending.

Value

data with tidyverse code attached

Author(s)

Owen Jin, Zhaoming Su

See Also

code

Examples

```
sorted <- sort_vars(iris,
    vars = c("Sepal.Width", "Sepal.Length"),
    asc = c(TRUE, FALSE)
)
cat(code(sorted))
head(sorted)
```

`standardize_vars` *Standardize the data of a numeric variable*

Description

Centre then divide by the standard error of the values in a numeric variable

Usage

```
standardize_vars(data, vars, names = NULL)
```

Arguments

`data`	a dataframe with the columns to standardize
`vars`	a character vector of the numeric variables in `data` to standardize
`names`	names for the created variables

Value

the original dataframe containing new columns of the standardized variables with tidyverse code attached

Author(s)

Zhaoming Su

See Also

code

Examples

```
standardized <- standardize_vars(iris, var = c("Sepal.Width", "Petal.Width"))
cat(code(standardized))
head(standardized)
```

survey_IQR *Interquartile range function for surveys*

Description

Calculates the interquartile range from complex survey data. A wrapper for taking differences of `svyquantile` at 0.25 and 0.75 quantiles, and meant to be called from within `summarize` (see srvyr package).

Usage

```
survey_IQR(x, na.rm = TRUE)
```

Arguments

`x`	A variable or expression
`na.rm`	logical, if `TRUE` missing values are removed

Value

a vector of interquartile ranges

Author(s)

Tom Elliott

Examples

```
library(survey)
library(srvyr)
data(api)

dstrata <- apistrat %>%
    as_survey(strata = stype, weights = pw)

dstrata %>%
    summarise(api99_iqr = survey_IQR(api99))
```

tidy_all_code *iNZight Tidy Code*

Description

Tidy code with correct indents and limit the code to the specific width

Usage

```
tidy_all_code(x, width = 80, indent = 4, outfile, incl_library = TRUE)
```

Arguments

x	character string or file name of the file containing messy code
width	the width of a line
indent	how many spaces for one indent
outfile	the file name of the file containing formatted code
incl_library	logical, if true, the output code will contain library name

Value

formatted code, optionally written to ‘outfile‘

Author(s)

Tom Elliott, Lushi Cai

transform_vars *Transform data of numeric variables*

Description

Transform the values of numeric variables by applying a mathematical function

Usage

```
transform_vars(data, vars, fn, names = NULL)
```

Arguments

data	a dataframe with the variables to transform
vars	a character of the numeric variables in data to transform
fn	the name (a string) of a valid R function
names	the names of the new variables

Value

the original dataframe containing the new columns of the transformed variable with tidyverse code attached

Author(s)

Zhaoming Su

See Also

code

Examples

```
transformed <- transform_vars(iris,
    var = "Petal.Length",
    fn = "log"
)
cat(code(transformed))
head(transformed)
```

---

validation_details *Details of Validation Rule Results*

Description

Generates the more detailed text required for the details section in `iNZValidateWin`.

Usage

```
validation_details(cf, v, var, id.var, df)
```

Arguments

`cf`	Confrontation object from `validate::confront()`
`v`	Validator that generated `cf`
`var`	Rule name to give details about
`id.var`	Variable name denoting a unique identifier for each observation
`df`	The dataset that was confronted

Value

A character vector giving each line of the summary detail text

Author(s)

Daniel Barnett

validation_summary *Validation Confrontation Summary*

Description

Generates a summary of a confrontation which gives basic information about each validation rule tested.

Usage

```
validation_summary(cf)
```

Arguments

cf	Confrontation object from `validate::confront()`

Value

A `data.frame` with number of tests performed, number of passes, number of failures, and failure percentage for each validation rule.

Author(s)

Daniel Barnett

vartype *Get variable type name*

Description

Get variable type name

Usage

```
vartype(x)
```

Arguments

x	vector to be examined

Value

character vector of the variable's type

Author(s)

Tom Elliott

`vartypes` *Get all variable types from data object*

Description

Get all variable types from data object

Usage

```
vartypes(x)
```

Arguments

`x`	data object (data.frame or inzdf)

Value

a named vector of variable types

Index