Package 'guideR'

November 26, 2025

Type Package

Title Miscellaneous Statistical Functions Used in 'guide-R'

Version 0.7.0

Description Companion package for the manual 'guide-R : Guide pour l'analyse de données d'enquêtes avec R' available at <https://larmarange.github.io/guide-R/>. 'guideR' implements miscellaneous functions introduced in 'guide-R' to facilitate statistical analysis and manipulation of survey data.

License GPL (>= 3)

URL https://larmarange.github.io/guideR/, https://github.com/larmarange/guideR

BugReports https://github.com/larmarange/guideR/issues

Depends R (>= 4.2)

Imports cli, dplyr (>= 1.1.0), forcats, ggplot2, labelled, lifecycle, pak, patchwork, purrr, renv, rlang, rstudioapi, scales, srvyr, stats, stringr, tidyr, tidyselect, utils

Encoding UTF-8

RoxygenNote 7.3.3

Suggests broom, broom.helpers, cardx, DT, FactoMineR, ggupset, ggstats, gt, gtsummary (>= 2.3.0), htmltools, htmlwidgets, nnet, parameters, spelling, survey, survival, testthat (>= 3.0.0)

Config/testthat/edition 3

Language en-US

NeedsCompilation no

Author Joseph Larmarange [aut, cre] (ORCID: <https://orcid.org/0000-0001-7097-700X>)

Maintainer Joseph Larmarange <joseph@larmarange.net>

Repository CRAN

Date/Publication 2025-11-26 12:30:02 UTC

Contents

`combine_answers` *Combine answers of a multiple answers question*

Description

Considering a multiple answers question coded as several binary variables (one per item), create a new variable (list column or character) combining all positive answers. If defined, use variable labels (see examples).

Usage

```
combine_answers(data, answers, into, value = NULL, sep = NULL)
```

Arguments

`data`	A data frame, data frame extension (e.g. a tibble), or a survey design object.
`answers`	<tidy-select> List of variables identifying the different answers of the question.
`into`	Names of new variables to create as character vector.

value	Value indicating a positive answer. By default, will use the maximum observed value and will display a message.
sep	An optional character string to separate the results and return a character. If NULL, return a list column (see examples).

Note

If NA is observed for at least one item, return NA.

Examples

```
d <-
  dplyr::tibble(
    q1a = sample(c("y", "n"), size = 200, replace = TRUE),
    q1b = sample(c("y", "n", "n", NA), size = 200, replace = TRUE),
    q1c = sample(c("y", "y", "n"), size = 200, replace = TRUE),
    q1d = sample("n", size = 200, replace = TRUE)
  )

d |> combine_answers(q1a:q1d, into = "combined")
d |> combine_answers(q1a:q1d, into = "combined", sep = ", ", value = "y")
d |> combine_answers(q1a:q1d, into = "combined", sep = " | ", value = "n")
```

---

cut_quartiles *Cut a continuous variable in quartiles*

Description

Convenient function to quickly cut a numeric vector into quartiles, i.e. by applying cut(x, breaks = fivenum(x)). Variable label is preserved by cut_quartiles().

Usage

```
cut_quartiles(x, include.lowest = TRUE, ...)
```

Arguments

x	a numeric vector which is to be converted to a factor by cutting.
include.lowest	logical, indicating if an 'x[i]' equal to the lowest (or highest, for right = FALSE) 'breaks' value should be included.
...	further arguments passed to base::cut().

Examples

```
mtcars$mpg |> cut_quartiles() |> summary()
```

grouped_tbl_pivot_wider *Helpers for grouped tables generated with* gtsummary

Description

A series of helpers for grouped tables generated by `tbl_regression()` in case of multinomial models, multi-components models or other grouped results. `grouped_tbl_pivot_wider()` allows to display results in a a wide format, with one set of columns per group. `multinom_add_global_p_pivot_wider()` is a specific case for multinomial models, when displaying global p-values in a wide format: it calls `gtsummary::add_global_p()`, followed by `grouped_tbl_pivot_wider()`, and then keep only the last column with p-values (see examples). Finally, as grouped regression tables doesn't have exactly the same structure as ungrouped tables, functions as `gtsummary::bold_labels()` do not always work properly. If the grouped table is kept in a long format, `style_grouped_tbl()` could be use to improve the output by styling variable labels, levels and/or group names. **TO BE NOTED:** to style group names, `style_grouped_tbl()` convert the table into a `gt` object with `gtsummary::as_gt()`. This function should therefore be used last. If the table is intended to be exported to another format, do not use `style_grouped_tbl()`.

Usage

```
grouped_tbl_pivot_wider(x)

multinom_add_global_p_pivot_wider(
  x,
  ...,
  p_value_header = "**Likelihood-ratio test**"
)

style_grouped_tbl(
  x,
  bold_groups = TRUE,
  uppercase_groups = TRUE,
  bold_labels = FALSE,
  italicize_labels = TRUE,
  indent_labels = 4L,
  bold_levels = FALSE,
  italicize_levels = FALSE,
  indent_levels = 8L
)
```

Arguments

x	A grouped regression table generated with `gtsummary::tbl_regression()`.
...	Additional arguments passed to `gtsummary::add_global_p()`.
p_value_header	Header for the p-value column.

bold_groups	Bold group group names?
uppercase_groups	Convert group names to upper case?
bold_labels	Bold variable labels?
italicize_labels	Italicize variable labels?
indent_labels	Number of spaces to indent variable labels.
bold_levels	Bold levels?
italicize_levels	Italicize levels?
indent_levels	Number of spaces to indent levels.

Value

A gtsummary or a gt table.

Examples

```
mod <- nnet::multinom(
  grade ~ stage + marker + age,
  data = gtsummary::trial,
  trace = FALSE
)
tbl <- mod |> gtsummary::tbl_regression(exponentiate = TRUE)
tbl
tbl |> grouped_tbl_pivot_wider()

tbl |> multinom_add_global_p_pivot_wider() |> gtsummary::bold_labels()
tbl |> style_grouped_tbl()
```

---

gtsummary_test *Additional tests for* gtsummary

Description

See gtsummary::tests for more details on how defining custom tests. fisher.simulate.p() implements Fisher test with computation of p-values by Monte Carlo simulation in larger than 2×2 tables (see stats::fisher.test()).

Usage

```
fisher.simulate.p(data, variable, by, ...)
```

Arguments

data	A data set.
variable	Name of the variable to test.
by	Name of the by variable.
...	Unused.

Examples

```
library(gtsummary)
trial |>
  tbl_summary(include = grade, by = trt) |>
  add_p(test = all_categorical() ~ "fisher.simulate.p")
```

---

gtsummary_themes *Themes for* gtsummary

Description

Additional themes for tables generated with gtsummary.

Usage

```
theme_gtsummary_prop_n(
  prop_stat = "{p}% ({n})",
  prop_digits = 1,
  mean_sd = FALSE,
  cont_digits = 1,
  set_theme = TRUE
)

theme_gtsummary_fisher_simulate_p(set_theme = TRUE)

theme_gtsummary_unweighted_n(
  n_unweighted_prefix = "",
  n_unweighted_suffix = " obs.",
  prop_digits = 1,
  mean_sd = FALSE,
  cont_digits = 1,
  overall_string = NULL,
  set_theme = TRUE
)

theme_gtsummary_bold_labels(set_theme = TRUE)
```

Arguments

`prop_stat`	(`character`) Statistics to display for categorical variables (see `gtsummary::tbl_summary()`).
`prop_digits`	(non-negative `integer`) Define the number of decimals to display for proportions.
`mean_sd`	(scalar `logical`) Also, set default summary statistics to mean and standard deviation in `gtsummary::tbl_summary()`. Default is `FALSE`.
`cont_digits`	(non-negative `integer`) Define the number of decimals to display for continuous variables.
`set_theme`	(scalar `logical`) Logical indicating whether to set the theme. Default is `TRUE`. When `FALSE` the named list of theme elements is returned invisibly
`n_unweighted_prefix, n_unweighted_suffix`	(`character`) Prefix and suffix displayed before and after the unweighted number of observations.
`overall_string`	(`character`) Optional string to name the *overall* column.

Details

`theme_gtsummary_prop_n()` displays, by default, proportions before the number of observations (between brackets). This function cannot be used simultaneously with `gtsummary::theme_gtsummary_mean_sd()`, but you can use the `mean_sd = TRUE` option of `theme_gtsummary_prop_n()`.

`theme_gtsummary_fisher_simulate_p()` modify the default test used for categorical variables by Fisher test, with computation of p-values by Monte Carlo simulation in larger than 2×2 tables.

`theme_gtsummary_unweighted_n()` modifies default values of tables returned by `gtsummary::tbl_svysummary()` and displays the unweighted number of observations instead of the weighted n.

`theme_gtsummary_bold_labels()` applies automatically `gtsummary::bold_labels()` to all tables generated with `gtsummary`.

Examples

```
library(gtsummary)

trial |>
  tbl_summary(include = c(grade, age), by = trt) |>
  add_p()

theme_gtsummary_prop_n(mean_sd = TRUE)
theme_gtsummary_fisher_simulate_p()
theme_gtsummary_bold_labels()
trial |>
  tbl_summary(include = c(grade, age), by = trt) |>
  add_p()
```

```
data("api", package = "survey")
apistrat$both[1:5] <- NA
apistrat |>
  srvyr::as_survey(strata = stype, weights = pw) |>
  tbl_svysummary(include = c(stype, both), by = awards) |>
  add_overall()

theme_gtsummary_unweighted_n()
apistrat |>
  srvyr::as_survey(strata = stype, weights = pw) |>
  tbl_svysummary(include = c(stype, both), by = awards) |>
  add_overall()


gtsummary::reset_gtsummary_theme()
```

`gtsummary_utilities` *Utilities for* `gtsummary`

Description

Utilities for tables generated with gtsummary.

Usage

```
bold_variable_group_headers(x)

italicize_variable_group_headers(x)

indent_levels(x, indent = 8L)

indent_labels(x, indent = 4L)
```

Arguments

x	A gtsummary object.
indent	An integer indicating how many space to indent text.

See Also

gtsummary::modify_bold(), gtsummary::modify_italic(), gtsummary::modify_indent()

Examples

```
library(gtsummary)
tbl <-
  trial |>
  tbl_summary(
    include = c(stage, grade, age, trt, response, death)
  ) |>
  add_variable_group_header(
    header = "Clinical situation at diagnosis",
    variables = c(stage, grade, age)
  ) |>
  add_variable_group_header(
    header = "Treatment and outcome",
    variables = c(trt, response, death)
  )
tbl

tbl |>
  bold_variable_group_headers() |>
  italicize_labels() |>
  indent_levels(indent = 8L)
```

`install_dependencies` *Install / Update project dependencies*

Description

This function uses `renv::dependencies()` to identify R package dependencies in a project and then calls `pak::pkg_install()` to install / update these packages. If some packages are not found, the function will install those available and returns a message indicated packages not installed/updated.

Usage

```
install_dependencies(dependencies = NULL, ask = TRUE)
```

Arguments

`dependencies`	An optional list of dependencies. If `NULL`, will be determined with `renv::dependencies()`. If equal to `"old"`, will use the list returned by `utils::old.packages()`.
`ask`	Whether to ask for confirmation when installing a different version of a package that is already installed. Installations that only add new packages never require confirmation.

Value

(Invisibly) A data frame with information about the installed package(s).

Examples

```
## Not run:
install_dependencies()

## End(Not run)
```

`is_different` *Comparison tests considering* `NA` *as values to be compared*

Description

`is_different()` and `is_equal()` performs comparison tests, considering `NA` values as legitimate values (see examples).

Usage

```
is_different(x, y)

is_equal(x, y)

cumdifferent(x)

num_cycle(x)
```

Arguments

`x`, `y`	Vectors to be compared.

Details

`cum_different()` allows to identify groups of continuous rows that have the same value. `num_cycle()` could be used to identify sub-groups that respect a certain condition (see examples).

`is_equal(x, y)` is equivalent to `(x == y & !is.na(x) & !is.na(y)) | (is.na(x) & is.na(y))`, and `is_different(x, y)` is equivalent to `(x != y & !is.na(x) & !is.na(y)) | xor(is.na(x), is.na(y))`.

Value

A vector of the same length as `x`.

Examples

```
v <- c("a", "b", NA)
is_different(v, "a")
is_different(v, NA)
is_equal(v, "a")
is_equal(v, NA)
d <- dplyr::tibble(group = c("a", "a", "b", "b", "a", "b", "c", "a"))
d |>
  dplyr::mutate(
    subgroup = cumdifferent(group),
    sub_a = num_cycle(group == "a")
  )
```

`leading_zeros` *Add leading zeros*

Description

Add leading zeros

Usage

```
leading_zeros(x, left_digits = NULL, digits = 0, prefix = "", suffix = "", ...)
```

Arguments

`x`	a numeric vector
`left_digits`	number of digits before decimal point, automatically computed if not provided
`digits`	number of digits after decimal point
`prefix, suffix`	Symbols to display before and after value
`...`	additional parameters passed to `base::formatC()`, as `big.mark` or `decimal.mark`

Value

A character vector of the same length as `x`.

See Also

`base::formatC()`, `base::sprintf()`

Examples

```
v <- c(2, 103.24, 1042.147, 12.4566, NA)
leading_zeros(v)
leading_zeros(v, digits = 1)
leading_zeros(v, left_digits = 6, big.mark = " ")
leading_zeros(c(0, 6, 12, 18), prefix = "M")
```

long_to_periods *Transform a data frame from long format to period format*

Description

Transform a data frame from long format to period format

Usage

```
long_to_periods(data, id, start, stop = NULL, by = NULL)
```

Arguments

data	A data frame, or a data frame extension (e.g. a tibble).
id	<tidy-select> Column containing individual ids
start	<tidy-select> Time variable indicating the beginning of each row
stop	<tidy-select> Optional time variable indicating the end of each row. If not provided, it will be derived from the dataset, considering that each row ends at the beginning of the next one.
by	<tidy-select> Co-variables to consider (optional)

Value

A tibble.

See Also

periods_to_long()

Examples

```
d <- dplyr::tibble(
  patient = c(1, 2, 3, 3, 4, 4, 4),
  begin = c(0, 0, 0, 1, 0, 36, 39),
  end = c(50, 6, 1, 16, 36, 39, 45),
  covar = c("no", "no", "no", "yes", "no", "yes", "yes")
)
d

d |> long_to_periods(id = patient, start = begin, stop = end)
d |> long_to_periods(id = patient, start = begin, stop = end, by = covar)

# If stop not provided, it is deduced.
# However, it considers that observation ends at the last start time.
d |> long_to_periods(id = patient, start = begin)
```

`observed_vs_theoretical` *Plot observed vs predicted distribution of a fitted model*

Description

Plot observed vs predicted distribution of a fitted model

Usage

```
observed_vs_theoretical(model)
```

Arguments

`model`	A statistical model.

Details

Has been tested with `stats::lm()` and `stats::glm()` models. It may work with other types of models, but without any warranty.

Value

A `ggplot2` plot.

Examples

```
# a linear model
mod <- lm(Sepal.Length ~ Sepal.Width + Species, data = iris)
mod |> observed_vs_theoretical()

# a logistic regression
mod <- glm(
  as.factor(Survived) ~ Class + Sex,
  data = titanic,
  family = binomial()
)
mod |> observed_vs_theoretical()
```

periods_to_long *Transform a data frame from period format to long format*

Description

Transform a data frame from period format to long format

Usage

```
periods_to_long(
  data,
  start,
  stop,
  time_step = 1,
  time_name = "time",
  keep = FALSE
)
```

Arguments

data	A data frame, or a data frame extension (e.g. a tibble).
start	<tidy-select> Time variable indicating the beginning of each row
stop	<tidy-select> Optional time variable indicating the end of each row. If not provided, it will be derived from the dataset, considering that each row ends at the beginning of the next one.
time_step	(numeric) Desired value for the time variable.
time_name	(character) Name of the time variable.
keep	(logical) Should start and stop variable be kept in the results?

Value

A tibble.

See Also

long_to_periods()

Examples

```
d <- dplyr::tibble(
  patient = c(1, 2, 3, 3),
  begin = c(0, 2, 0, 3),
  end = c(6, 4, 2, 8),
  covar = c("no", "yes", "no", "yes")
)
d
```

```
d |> periods_to_long(start = begin, stop = end)
d |> periods_to_long(start = begin, stop = end, time_step = 5)
```

`plot_inertia_from_tree` *Plot inertia, absolute loss and relative loss from a classification tree*

Description

Plot inertia, absolute loss and relative loss from a classification tree

Usage

```
plot_inertia_from_tree(tree, k_max = 15)

get_inertia_from_tree(tree, k_max = 15)
```

Arguments

`tree`	A dendrogram, i.e. an stats::hclust object, an FactoMineR::HCPC object or an object that can be converted to an stats::hclust object with `stats::as.hclust()`.
`k_max`	Maximum number of clusters to return / plot.

Value

A `ggplot2` plot or a tibble.

Examples

```
hc <- hclust(dist(USArrests))
get_inertia_from_tree(hc)
plot_inertia_from_tree(hc)
```

`plot_multiple_answers` *Plot a multiple answers question*

Description

Considering a multiple answers question coded as several binary variables (one per answer), plot the proportion of positive answers. If `combine_answers = FALSE`, plot the proportion of positive answers of each item, separately. If `combine_answers = FALSE`, combine the different answers (see `combine_answers()`) and plot the proportion of each combination (`ggupset` package required when `flip = FALSE`). See `proportion()` for more details on the way proportions and confidence intervals are computed. By default, return a bar plot, but other geometries could be used (see examples). If defined, use variable labels (see examples).

Usage

```
plot_multiple_answers(
  data,
  answers = dplyr::everything(),
  value = NULL,
  by = NULL,
  combine_answers = FALSE,
  combine_sep = " | ",
  missing_label = " missing",
  none_label = "none",
  drop_na = FALSE,
  sort = c("none", "ascending", "descending", "degrees"),
  geom = "bar",
  ...,
  show_ci = TRUE,
  conf_level = 0.95,
  ci_color = "black",
  show_labels = TRUE,
  labels_labeller = scales::label_percent(1),
  labels_size = 3.5,
  labels_color = "black",
  flip = FALSE,
  return_data = FALSE
)

plot_multiple_answers_dodge(
  data,
  answers = dplyr::everything(),
  value = NULL,
  by,
  combine_answers = FALSE,
  combine_sep = " | ",
  missing_label = " missing",
  none_label = "none",
  drop_na = FALSE,
  sort = c("none", "ascending", "descending", "degrees"),
  geom = c("bar", "point"),
  width = 0.75,
  ...,
  show_ci = TRUE,
  conf_level = 0.95,
  ci_color = "black",
  show_labels = TRUE,
  labels_labeller = scales::label_percent(1),
  labels_size = 3.5,
  labels_color = "black",
  flip = FALSE
)
```

Arguments

data	A data frame, data frame extension (e.g. a tibble), or a survey design object.
answers	<tidy-select> List of variables identifying the different answers of the question.
value	Value indicating a positive answer. By default, will use the maximum observed value and will display a message.
by	<tidy-select> Optional list of variables to compare (using facets).
combine_answers	Should answers be combined? (see examples)
combine_sep	Character string to separate combined answers.
missing_label	When combining answers and drop_na = FALSE, label for missing values.
none_label	When combining answers and flip = TRUE, label when no item is selected.
drop_na	Should any observation with a least one NA value be dropped?
sort	Should answers be sorted according to their proportion? They could also be sorted by degrees (number of elements) when combining answers.
geom	Geometry to use for plotting proportions ("bar" by default).
...	Additional arguments passed to the geom defined by geom.
show_ci	Display confidence intervals?
conf_level	Confidence level for the confidence intervals.
ci_color	Color of the error bars representing confidence intervals.
show_labels	Display proportion labels?
labels_labeller	Labeller function for proportion labels.
labels_size	Size of proportion labels.
labels_color	Color of proportion labels.
flip	Flip x and y axis?
return_data	Return computed data instead of the plot?
width	Dodging width.

Note

If drop_na = TRUE, any observation with at least one NA value for one item will be dropped. If drop_na = FALSE and combine_answers = FALSE, NA values for a specific answer are excluded the denominator when computing proportions. Therefore, all proportions may be computed on different population sizes. If drop_na = FALSE and combine_answers = TRUE, any observation with at least one NA value will be labeled with missing_label.

Examples

```
d <-
  dplyr::tibble(
    q1a = sample(c("y", "n"), size = 200, replace = TRUE),
    q1b = sample(c("y", "n", "n", NA), size = 200, replace = TRUE),
    q1c = sample(c("y", "y", "n"), size = 200, replace = TRUE),
    q1d = sample("n", size = 200, replace = TRUE)
  )

d |> plot_multiple_answers(q1a:q1c)

d |>
  labelled::set_variable_labels(
    q1a = "apple",
    q1b = "banana",
    q1c = "chocolate",
    q1d = "Dijon mustard"
  ) |>
  plot_multiple_answers(
    value = "y",
    drop_na = TRUE,
    sort = "desc",
    fill = "lightblue",
    flip = TRUE
  )


d |>
  plot_multiple_answers(
    combine_answers = TRUE,
    value = "y",
    fill = "#DDCC77",
    drop_na = TRUE
  )

d |>
  plot_multiple_answers(
    combine_answers = TRUE,
    value = "y",
    flip = TRUE,
    mapping = ggplot2::aes(fill = prop),
    show.legend = FALSE
  ) +
  ggplot2::scale_fill_distiller(palette = "Spectral")

d$group <- sample(c("group A", "group B"), size = 200, replace = TRUE)
d |>
  plot_multiple_answers(
    answers = q1a:q1d,
    by = group,
    combine_answers = TRUE,
    sort = "degrees",
```

```
    value = "y",
    fill = "grey80"
  )

d |>
  plot_multiple_answers_dodge(q1a:q1d, by = group)
d |>
  plot_multiple_answers_dodge(q1a:q1d, by = group, flip = TRUE)
d |>
  plot_multiple_answers_dodge(q1a:q1d, by = group, combine_answers = TRUE)
```

`plot_proportions` *Plot proportions by sub-groups*

Description

Plot one or several proportions (defined by logical conditions) by sub-groups. See `proportion()` for more details on the way proportions and confidence intervals are computed. By default, return a bar plot, but other geometries could be used (see examples). `stratified_by()` is an helper function facilitating a stratified analyses (i.e. proportions by groups stratified according to a third variable, see examples). `dummy_proportions()` is an helper to easily convert a categorical variable into dummy variables and therefore showing the proportion of each level of the original variable (see examples).

Usage

```
plot_proportions(
  data,
  condition,
  by = NULL,
  drop_na_by = FALSE,
  convert_continuous = TRUE,
  geom = "bar",
  ...,
  show_overall = TRUE,
  overall_label = "Overall",
  show_ci = TRUE,
  conf_level = 0.95,
  ci_color = "black",
  show_pvalues = TRUE,
  pvalues_test = c("fisher", "chisq"),
  pvalues_labeller = scales::label_pvalue(add_p = TRUE),
  pvalues_size = 3.5,
  show_labels = TRUE,
```

```
  labels_labeller = scales::label_percent(1),
  labels_size = 3.5,
  labels_color = "black",
  show_overall_line = FALSE,
  overall_line_type = "dashed",
  overall_line_color = "black",
  overall_line_width = 0.5,
  facet_labeller = ggplot2::label_wrap_gen(width = 50, multi_line = TRUE),
  flip = FALSE,
  free_scale = FALSE,
  return_data = FALSE
)

stratified_by(condition, strata)

dummy_proportions(variable)
```

Arguments

data	A data frame, data frame extension (e.g. a tibble), or a survey design object.
condition	<data-masking> A condition defining a proportion, or a dplyr::tibble() defining several proportions (see examples).
by	<tidy-select> List of variables to group by (comparison is done separately for each variable).
drop_na_by	Remove NA values in by variables?
convert_continuous	Should continuous variables (with 5 unique values or more) be converted to quartiles (using cut_quartiles())?
geom	Geometry to use for plotting proportions ("bar" by default).
...	Additional arguments passed to the geom defined by geom.
show_overall	Display "Overall" column?
overall_label	Label for the overall column.
show_ci	Display confidence intervals?
conf_level	Confidence level for the confidence intervals.
ci_color	Color of the error bars representing confidence intervals.
show_pvalues	Display p-values in the top-left corner?
pvalues_test	Test to compute p-values for data frames: "fisher" for stats::fisher.test() (with simulate.p.value = TRUE) or "chisq" for stats::chisq.test(). Has no effect on survey objects for those survey::svychisq() is used.
pvalues_labeller	Labeller function for p-values.
pvalues_size	Text size for p-values.
show_labels	Display proportion labels?

labels_labeller
: Labeller function for proportion labels.

labels_size
: Size of proportion labels.

labels_color
: Color of proportion labels.

show_overall_line
: Add an overall line?

overall_line_type
: Line type of the overall line.

overall_line_color
: Color of the overall line.

overall_line_width
: Line width of the overall line.

facet_labeller
: Labeller function for strip labels.

flip
: Flip x and y axis?

free_scale
: Allow y axis to vary between conditions?

return_data
: Return computed data instead of the plot?

strata
: Stratification variable

variable
: Variable to be converted into dummy variables.

Examples

```
titanic |>
  plot_proportions(
    Survived == "Yes",
    overall_label = "All",
    labels_color = "white"
  )

titanic |>
  plot_proportions(
    Survived == "Yes",
    by = c(Class, Sex),
    fill = "lightblue"
  )



titanic |>
  plot_proportions(
    Survived == "Yes",
    by = c(Class, Sex),
    fill = "lightblue",
    flip = TRUE
  )

titanic |>
  plot_proportions(
    Survived == "Yes",
```

```
    by = c(Class, Sex),
    geom = "point",
    color = "red",
    size = 3,
    show_labels = FALSE
  )

titanic |>
  plot_proportions(
    Survived == "Yes",
    by = c(Class, Sex),
    geom = "area",
    fill = "lightgreen",
    show_overall = FALSE
  )

titanic |>
  plot_proportions(
    Survived == "Yes",
    by = c(Class, Sex),
    geom = "line",
    color = "purple",
    ci_color = "darkblue",
    show_overall = FALSE
  )

titanic |>
  plot_proportions(
    Survived == "Yes",
    by = -Survived,
    mapping = ggplot2::aes(fill = variable),
    color = "black",
    show.legend = FALSE,
    show_overall_line = TRUE,
    show_pvalues = FALSE
 )

# defining several proportions

titanic |>
  plot_proportions(
    dplyr::tibble(
      Survived = Survived == "Yes",
      Male = Sex == "Male"
    ),
    by = c(Class),
    mapping = ggplot2::aes(fill = condition)
  )

titanic |>
  plot_proportions(
    dplyr::tibble(
      Survived = Survived == "Yes",
```

```
      Male = Sex == "Male"
    ),
    by = c(Class),
    mapping = ggplot2::aes(fill = condition),
    free_scale = TRUE
  )

iris |>
  plot_proportions(
    dplyr::tibble(
      "Long sepal" = Sepal.Length > 6,
      "Short petal" = Petal.Width < 1
    ),
    by = Species,
    fill = "palegreen"
  )

iris |>
  plot_proportions(
    dplyr::tibble(
      "Long sepal" = Sepal.Length > 6,
      "Short petal" = Petal.Width < 1
    ),
    by = Species,
    fill = "palegreen",
    flip = TRUE
  )

# works with continuous by variables
iris |>
  labelled::set_variable_labels(
    Sepal.Length = "Length of the sepal"
  ) |>
  plot_proportions(
    Species == "versicolor",
    by = dplyr::contains("leng"),
    fill = "plum",
    colour = "plum4"
  )

# works with survey object
titanic |>
  srvyr::as_survey() |>
  plot_proportions(
    Survived == "Yes",
    by = c(Class, Sex),
    fill = "darksalmon",
    color = "black",
    show_overall_line = TRUE,
    labels_labeller = scales::label_percent(.1)
 )
```

```
# stratified analysis
titanic |>
  plot_proportions(
    (Survived == "Yes") |>  stratified_by(Sex),
    by = Class,
    mapping = ggplot2::aes(fill = condition)
  ) +
  ggplot2::theme(legend.position = "bottom") +
  ggplot2::labs(fill = NULL)

# Convert Class into dummy variables
titanic |>
  plot_proportions(
    dummy_proportions(Class),
    by = Sex,
    mapping = ggplot2::aes(fill = level)
  )
```

`plot_trajectories` *Plot trajectories*

Description

Create a trajectory index plot (similar to sequence index plot) from a data frame in long or period format.

Usage

```
plot_trajectories(
  data,
  id,
  time,
  fill,
  by = NULL,
  sort_by = NULL,
  nudge_x = NULL,
  hide_y_labels = NULL,
  facet_labeller = ggplot2::label_wrap_gen(width = 50, multi_line = TRUE),
  ...
)

plot_periods(
  data,
  id,
  start,
  stop,
  fill,
  by = NULL,
```

```
  sort_by = NULL,
  nudge_x = NULL,
  hide_y_labels = NULL,
  facet_labeller = ggplot2::label_wrap_gen(width = 50, multi_line = TRUE),
  ...
)
```

Arguments

data	A data frame, or a data frame extension (e.g. a tibble).
id	<tidy-select> Column containing individual ids.
time	<tidy-select> Time variable.
fill	<tidy-select> Variable mapped to fill aesthetic.
by	<tidy-select> Optional variables to group by.
sort_by	<tidy-select> Optional variables to sort trajectories.
nudge_x	Optional amount of horizontal distance to move.
hide_y_labels	Hide y labels? If NULL, hide them when more than 20 trajectories are displayed.
facet_labeller	Labeller function for strip labels.
...	Additional arguments passed to ggplot2::geom_tile()
start, stop	<tidy-select> Start and stop variables of the periods.

Note

plot_trajectories() assumes that data are stored in a long format (i.e. one row per unit of time). You can use tidyr::pivot_longer() or periods_to_long() to transform your data in such format. By default, tiles are centered on the value of time. You can adjust horizontal position with nudge_x. By default, each row is assumed to represent one unit of time and represented with a width of 1. You can adjust tiles' width with width.

plot_periods() is adapted for period format with a start and a stop variable. You can use long_to_periods() to transform your data in such format. Beginning and ending of each tile is determined by start and stop arguments.

Examples

```
d <- dplyr::tibble(
  id = c(1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3, 3),
  time = c(0:3, 0:2, 0:4),
  status = c("a", "a", "b", "b", "b", "b", "a", "b", "b", "b", "b", "a"),
  group = c("f", "f", "f", "f", "f", "f", "f", "m", "m", "m", "m", "m")
)

d |> plot_trajectories(id = id, time = time, fill = status, colour = "black")
d |> plot_trajectories(id = id, time = time, fill = status, nudge_x = .5)
d |> plot_trajectories(id = id, time = time, fill = status, by = group)

d2 <- d |>
  dplyr::mutate(end = time + 1) |>
```

```
  long_to_periods(id = id, start = time, stop = end, by = status)
d2
d2 |> plot_periods(
  id = id,
  start = time,
  stop = end,
  fill = status,
  colour = "black",
  height = 0.8
)
```

---

proportion *Compute proportions*

Description

proportion() lets you quickly count observations (like dplyr::count()) and compute relative proportions. Proportions are computed separately by group (see examples).

Usage

```
proportion(data, ...)

## S3 method for class 'data.frame'
proportion(
  data,
  ...,
  .by = NULL,
  .na.rm = FALSE,
  .weight = NULL,
  .scale = 100,
  .sort = FALSE,
  .drop = FALSE,
  .drop_na_by = FALSE,
  .conf.int = FALSE,
  .conf.level = 0.95,
  .options = list(correct = TRUE)
)

## S3 method for class 'survey.design'
proportion(
  data,
  ...,
  .by = NULL,
  .na.rm = FALSE,
  .scale = 100,
  .sort = FALSE,
  .drop_na_by = FALSE,
```

```
  .conf.int = FALSE,
  .conf.level = 0.95,
  .options = NULL
)

## Default S3 method:
proportion(
  data,
  ...,
  .na.rm = FALSE,
  .scale = 100,
  .sort = FALSE,
  .drop = FALSE,
  .conf.int = FALSE,
  .conf.level = 0.95,
  .options = list(correct = TRUE)
)
```

Arguments

`data`	A vector, a data frame, data frame extension (e.g. a tibble), or a survey design object.
`...`	<`data-masking`> Variable(s) for those computing proportions.
`.by`	<`tidy-select`> Optional additional variables to group by (in addition to those eventually previously declared using `dplyr::group_by()`).
`.na.rm`	Should `NA` values be removed (from variables declared in `...`)?
`.weight`	<`data-masking`> Frequency weights. Can be `NULL` or a variable.
`.scale`	A scaling factor applied to proportion. Use 1 for keeping proportions unchanged.
`.sort`	If `TRUE`, will show the highest proportions at the top.
`.drop`	If `TRUE`, will remove empty groups from the output.
`.drop_na_by`	If `TRUE`, will remove any `NA` values observed in the `.by` variables (or variables defined with `dplyr::group_by()`).
`.conf.int`	If `TRUE`, will estimate confidence intervals.
`.conf.level`	Confidence level for the returned confidence intervals.
`.options`	Additional arguments passed to `stats::prop.test()` or `srvyr::survey_prop()`.

Value

A tibble.

A tibble with one row per group.

Examples

```
# using a vector
titanic$Class |> proportion()
```

```
# univariable table
titanic |> proportion(Class)
titanic |> proportion(Class, .sort = TRUE)
titanic |> proportion(Class, .conf.int = TRUE)
titanic |> proportion(Class, .conf.int = TRUE, .scale = 1)

# bivariable table
titanic |> proportion(Class, Survived) # proportions of the total
titanic |> proportion(Survived, .by = Class) # row proportions
titanic |> # equivalent syntax
  dplyr::group_by(Class) |>
  proportion(Survived)

# combining 3 variables or more
titanic |> proportion(Class, Sex, Survived)
titanic |> proportion(Sex, Survived, .by = Class)
titanic |> proportion(Survived, .by = c(Class, Sex))

# missing values
dna <- titanic
dna$Survived[c(1:20, 500:530)] <- NA
dna |> proportion(Survived)
dna |> proportion(Survived, .na.rm = TRUE)


## SURVEY DATA ----------------------------------------------------

ds <- srvyr::as_survey(titanic)

# univariable table
ds |> proportion(Class)
ds |> proportion(Class, .sort = TRUE)
ds |> proportion(Class, .conf.int = TRUE)
ds |> proportion(Class, .conf.int = TRUE, .scale = 1)

# bivariable table
ds |> proportion(Class, Survived) # proportions of the total
ds |> proportion(Survived, .by = Class) # row proportions
ds |> dplyr::group_by(Class) |> proportion(Survived)

# combining 3 variables or more
ds |> proportion(Class, Sex, Survived)
ds |> proportion(Sex, Survived, .by = Class)
ds |> proportion(Survived, .by = c(Class, Sex))

# missing values
dsna <- srvyr::as_survey(dna)
dsna |> proportion(Survived)
dsna |> proportion(Survived, .na.rm = TRUE)
```

`round_preserve_sum` *Round values while preserve their rounded sum in R*

Description

Sometimes, the sum of rounded numbers (e.g., using `base::round()`) is not the same as their rounded sum.

Usage

```
round_preserve_sum(x, digits = 0)
```

Arguments

`x`	Numerical vector to sum.
`digits`	Number of decimals for rounding.

Details

This solution applies the following algorithm

- Round down to the specified number of decimal places
- Order numbers by their remainder values
- Increment the specified decimal place of values with k largest remainders, where k is the number of values that must be incremented to preserve their rounded sum

Value

A numerical vector of same length as `x`.

Source

https://biostatmatt.com/archives/2902

Examples

```
sum(c(0.333, 0.333, 0.334))
round(c(0.333, 0.333, 0.334), 2)
sum(round(c(0.333, 0.333, 0.334), 2))
round_preserve_sum(c(0.333, 0.333, 0.334), 2)
sum(round_preserve_sum(c(0.333, 0.333, 0.334), 2))
```

`step_with_na` *Apply* `step()`*, taking into account missing values*

Description

When your data contains missing values, concerned observations are removed from a model. However, then at a later stage, you try to apply a descending stepwise approach to reduce your model by minimization of AIC, you may encounter an error because the number of rows has changed.

Usage

```
step_with_na(model, ...)

## Default S3 method:
step_with_na(model, ..., full_data = eval(model$call$data))

## S3 method for class 'svyglm'
step_with_na(model, ..., design)
```

Arguments

`model`	A model object.
`...`	Additional parameters passed to `stats::step()`.
`full_data`	Full data frame used for the model, including missing data.
`design`	Survey design previously passed to `survey::svyglm()`.

Details

`step_with_na()` applies the following strategy:

- recomputes the models using only complete cases;
- applies `stats::step()`;
- recomputes the reduced model using the full original dataset.

`step_with_na()` has been tested with `stats::lm()`, `stats::glm()`, `nnet::multinom()`, `survey::svyglm()` and `survival::coxph()`. It may be working with other types of models, but with no warranty.

In some cases, it may be necessary to provide the full dataset initially used to estimate the model.

`step_with_na()` may not work inside other functions. In that case, you may try to pass `full_data` to the function.

Value

The stepwise-selected model.

Examples

```
set.seed(42)
d <- titanic |>
  dplyr::mutate(
    Group = sample(
      c("a", "b", NA),
      dplyr::n(),
      replace = TRUE
    )
  )
mod <- glm(as.factor(Survived) ~ ., data = d, family = binomial())
# step(mod) should produce an error
mod2 <- step_with_na(mod, full_data = d)
mod2


## WITH SURVEY ----------------------------------------

library(survey)
ds <- d |>
  dplyr::mutate(Survived = as.factor(Survived)) |>
  srvyr::as_survey()
mods <- survey::svyglm(
  Survived ~ Class + Group + Sex,
  design = ds,
  family = quasibinomial()
)
mod2s <- step_with_na(mods, design = ds)
mod2s
```

---

titanic *Titanic data set in long format*

Description

This `titanic` dataset is equivalent to `datasets::Titanic |> dplyr::as_tibble() |> tidyr::uncount(n)`.

Usage

```
titanic
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 2201 rows and 4 columns.

See Also

datasets::Titanic

unrowwise *Remove row-wise grouping*

Description

Remove row-wise grouping created with `dplyr::rowwise()` while preserving any other grouping declared with `dplyr::group_by()`.

Usage

```
unrowwise(data)
```

Arguments

data	A tibble.

Value

A tibble.

Examples

```
titanic |> dplyr::rowwise()
titanic |> dplyr::rowwise() |> unrowwise()

titanic |> dplyr::group_by(Sex, Class) |> dplyr::rowwise()
titanic |> dplyr::group_by(Sex, Class) |> dplyr::rowwise() |> unrowwise()
```

view_dictionary *Display the variable dictionary of a data frame in the RStudio viewer*

Description

Generates an interactive variable dictionary based on `labelled::look_for()`. Accepts data frames, tibbles, and also survey objects.

Usage

```
view_dictionary(data = NULL, details = c("basic", "none", "full"))

view_detailed_dictionary(data = NULL)

to_DT(
  x,
  caption = NULL,
 column_labels = list(pos = "#", variable = "Variable", col_type = "Type", label =
```

```
    "Variable label", values = "Values", missing = "Missing values", unique_values =
     "Unique values", na_values = "User-defined missings (values)", na_range =
     "User-defined missings (range)")
)
```

Arguments

data	a data frame, a tibble or a survey object (if NULL, will use the text you currently select in **RStudio**, useful if the function is called through the corresponding addin)
details	add details about each variable (see labelled::look_for())
x	a tibble returned by look_for()
caption	an optional caption for the table
column_labels	Optional column labels

Details

view_dictionary() calls labelled::look_for() and applies to_DT() to the result to produce an HTML version of the variable dictionary. If you are using **RStudio**, it will be displayed by default in the *Viewer* pane, allowing to have the dictionary close to your code.

view_detailed_dictionary() is similar to view_dictionary() with the option details = "full".

These two functions are also available through dedicated addins in **RStudio**. To use them, select the name of a data frame, then choose *View variable dictionary* in the *Addins* menu.

Note

to_DT() is an utility to convert the result of labelled::look_for() into a DT::datatable().

Examples

```
iris |> view_dictionary()


iris |> labelled::look_for(details = TRUE) |> to_DT()
```

Index