Package 'guideR'

August 20, 2026

Type Package

Title Miscellaneous Statistical Functions Used in 'guide-R'

Version 0.12.0

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

Encoding UTF-8

Suggests broom, broom.mixed, car, cardx, dominanceanalysis, DT,
FactoMineR, ggupset, ggstats, gt, gtsummary (>= 2.5.0),
htmltools, htmlwidgets, khroma, MAIHDA (>= 0.2.1), MASS, nnet,
parameters, performance, spelling, survey, survival, testthat
(>= 3.0.0), TraMineR, vdiffr, WeMix

Config/testthat/edition 3

Language en-US

Config/roxygen2/version 8.1.0

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 2026-08-20 14:10:15 UTC

Contents

add_interactions_by_step . . . 3
combine_answers . . . 3
contributions . . . 4
cut_quartiles . . . 9
da.svyglm.fit . . . 10
deprecated . . . 10
grouped_tbl_pivot_wider . . . 11
gtsummary_test . . . 13
gtsummary_themes . . . 14
gtsummary_utilities . . . 16
install_dependencies . . . 17
is_different . . . 18
leading_zeros . . . 19
long_to_periods . . . 20
long_to_seq . . . 21
mean_sd . . . 23
median_iqr . . . 25
observed_vs_theoretical . . . 27
periods_to_long . . . 28
plot_categorical . . . 29
plot_continuous . . . 31
plot_inertia_from_tree . . . 34
plot_means . . . 35
plot_multiple_answers . . . 38
plot_proportions . . . 41
plot_trajectories . . . 46
proportion . . . 49
round_preserve_sum . . . 51
safe_pal . . . 52
step_with_na . . . 54
svyoneway . . . 55
tbl_maihda . . . 56
titanic . . . 64
unrowwise . . . 64
view_dictionary . . . 65

Index **67**

add_interactions_by_step *Add potential relevant interactions using* `step()`

Description

[Experimental] Add potential relevant interactions to a model using `stats::step()`. The function extract the formula of the model, identifies all potential interactions and pass them as the **upper** component of the `scope` argument to `stats::step()`. The current model formula is passed as the **lower** component of `scope`.

Usage

```
add_interactions_by_step(model, ...)

## Default S3 method:
add_interactions_by_step(model, ...)
```

Arguments

`model`	A model object.
`...`	Additional parameters passed to `stats::step()`.

Value

The stepwise-selected model.

Examples

```
mod <- glm(as.factor(Survived) ~ ., data = titanic, family = binomial())
mod |> add_interactions_by_step()
```

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")

# works with survey objects
d |>
  srvyr::as_survey() |>
  combine_answers(q1a:q1d, into = "combined")
```

---

contributions *Variable contribution and Dominance analysis*

Description

[Experimental] Use an analysis of variance or an analysis of deviance to estimate the contribution of each variable in reducing variance or deviance (sort of R2 decomposition). This contribution could be explained as a proportion of the total deviance (*total contribution*), of the residual deviance (*partial contribution*) or of the explained deviance (*relative contribution*), see details for more information. contributions() computes the different contributions. tbl_contributions() displays the results as a formatted gt table, taking into account variable labels. Alternatively, tbl_domimance() could be used to perform a dominance analysis with dominanceanalysis::dominanceAnalysis() and display the results as a formatted gt table.

Usage

```
contributions(mod, type = c("II", "III", "I", 1, 2, 3, "drop1", "add1"), ...)

tbl_contributions(
  mod,
  type = c("II", "III", "I", 1, 2, 3, "drop1", "add1"),
  ...,
  show = c("Total", "Partial", "Relative"),
  decimals = 1,
  notes = TRUE,
  lang = c("en", "fr")
)

total_deviance(mod)

tbl_dominance(
  mod,
  indice = NULL,
  decimals = 1,
  totals = TRUE,
  notes = TRUE,
  lang = c("en", "fr")
)
```

Arguments

mod	a statistical model
type	type of Anova, Roman numerals being equivalent to the corresponding Arabic numerals: `stats::anova()` will be used for type-I, Anova, `car::Anova()` for types II and III; an alternative method, `"drop1"` uses `stats::drop1()` to calculate the reduction of deviance associated with each predictor (in the absence of interactions, is equivalent to a type-II Anova); `"add1"` uses `stats::add1()` to calculate the reduction of deviance associated with each predictor in an univariable model (model with just this predictor compared to the null model), in this scenario, partial and relative contribution are not defined.
...	additional parameters passed to `stats::anova()`, `car::Anova()`, `stats::drop1()` or `stats::add1()`
show	list of contributions to display
decimals	number of decimals for deviance and contributions
notes	should table notes be added?
lang	language used for column labels and notes
indice	fit indice name, see `dominanceanalysis::dominanceAnalysis()`
totals	add a summary row with totals

Details

In linear regression, the squared multiple correlation, R2 is used to assess goodness of fit as it represents the proportion of variance in the criterion that is explained by the predictors. It could be expressed as `1 - RSS / TSS` where `TSS` represents the total sum of squares (the overall observed variance) and `RSS` represents the residual sum of squares (i.e. the variance not explained by the full model).

For generalized linear models (GLM), model fitting does not rely on ordinary least squares (OLS). It is achieved by maximum likelihood. Therefore, sum of squares is not available and R2 cannot be computed. An alternative goodness of fit measure used in such case is pseudo R2. The' McFadden pseudo R2 (sometimes called likelihood ratio index) could be expressed using deviance rather than likelihood. In such case, it is equal to `1 - Dr / Dt` where `Dt` represents the full deviance (i.e. the deviance of the null model, a model with no predictor) and `Dr` the residual deviance (i.e. the deviance of the full model). The McFadden pseudo R2 corresponds the proportion of deviance reduced by the model. Deviance is a generalization of the idea of using the sum of squares of residuals in the cases where model-fitting is achieved by maximum likelihood.

In **R**, the residual deviance (`Dr`) of a GLM could be obtain with `stats::deviance()`. For a linear model, `stats::deviance()` reports RSS. The function `total_deviance()` could be used to get the total deviance (`Dt`), i.e. the deviance of the null model. For a linear model, it provides TSS.

Analysis of variance (Anova) is common to identify if the different predictors have a significant effect on a model. Anova approaches have been extended to also covers *analysis of deviance*. In the context of Anova-like tests, it is common to report effect sizes indicators representing the amount of variance or deviance explained by each variable included in the model. Such approach is also known as *R2* decomposition. These indicators are based on `Dpred` (the delta of deviance) or `SSpred` (the delta of the sum of squares, i.e. the delta of variance) reduced by the inclusion of a specific predictor in the model.

A first measure, known as *Eta-squared* in the context of linear models, expresses this delta of sum of squares (variance) as a proportion of the total variance (`SSpred / TSS`). In the context of GLMs, the delta of deviance could similarly expressed as a proportion of the total deviance (`Dpred / Dt`), also known as the *semi-partial pseudo-R2*. This indicator represents the **total contribution** of a predictor in the reduction of deviance.

An alternative, known as *partial Eta-squared*, in the context of linear models, could be expressed as `SSpred / (RSS + SSpred)`, the variance uniquely explained by the effect expressed as the proportion of variance not explained by the other effects. Here the variance explained by the other effects in the model is completely partialed out. Similarly, for GLMs, *partial pseudo-R2* is expressed as `Dpred / (Dr + Dpred)`, where `Dr` represents the residual deviance of the full model. This **partial contribution** is the proportion of partial variance/deviance uniquely explained by the associated effect.

Finally, it is possible to express a **relative contribution** as the proportion of the variance/deviance explained by the model, that could be be expressed as `TSS - RSS` or `Dt - Dr`. Therefore, relative contribution is equal to `SSpred / (TSS - RSS)` or `Dpred / (Dt - Dr)` and represents the relative reduction of variance/deviance of the predictor compared to the total reduction of the variance / deviance by the full model.

It is crucial to understand the different types of Anova.

In a type-I Anova, as performed by `stats::anova()`, the different predictors are included sequentially and in-order into the model. Such analysis is therefore order-dependant. The effect of a

predictor is computed once taken into account the previous predictors (but not the other one introduced later). The first factor is tested without adjustment. The second factor is tested after removing the effect of the first. The third is tested after removing the effect of the first and the second. The advantage of such approach is that the sum of relative contributions is equal to 100%. However, the contribution of each variable changes if the variables are entered in a different order.

Type II-Anova (default of `car::Anova()`) tests each main effect adjusted for all other effects of the same order or lower, but not for interactions involving that factor. Each main effect is tested as if it were the last main effect entered. Type II are order-independent for main effects and are generally preferred when there is no interaction. They have higher power than Type III for testing main effects because they do not adjust for the interaction term.

Type III-Anova (also done with `car::Anova()`) tests each effect adjusted for all other effects in the model, including higher-order interactions. Each effect is tested as if it were the last one entered into a model containing all other effects. Type III are order-independent. They require a specific contrast coding (typically sum-to-zero or Helmert) to be interpretable.

`car::Anova()` calculates type-II or type-III Anova tables indicating for each predictor the variance or deviance explained by adding this predictor in the model compared to a model without this predictor (but keeping all other predictors in the model).

`contributions()` also includes two alternatives: `"drop1"` and `"add1"`.

`type = "drop1"` uses `stats::drop1()` which is equivalent, in the absence, of interaction terms to a type-II Anova. For some models (such as `survey::svyglm()` models), it provides better estimates than `car::Anova()`.

`type = "add1"` uses `stats::add1()` to calculate the reduction of deviance associated with each predictor in an univariable model (a model with just this predictor compared to the null model). In this scenario, partial and relative contribution are not defined. It provides an estimate of the contribution of a variable in the absence of all other predictors.

For `survey::svyglm()` object, only type II and III are supported. The `type = "drop1"` is recommended, in the absence of interactions, for such models.

In a type-II or type-III Anova, the sum of relative contributions would equal 100% only if all predictors are perfectly independent. In practice, the sum is never equal to 100% due to some correlation between predictors.

Some approaches, such as dominance analysis, have been developed for decomposing the coefficient of determination (R2) into individual contributions of predictors in regression models, accounting for correlations between predictors. They address a key challenge in relative importance analysis: how to allocate shared variance when predictors are collinear.

Dominance analysis follows a systematic, computationally intensive approach based on all-subsets regression. Models of every possible combination of predictors are generated, each pair of predictors are compared by examining their incremental contributions within matching subsets. Finally, the dominance relationship is determined based on three levels: complete dominance (variable A always contributes more), conditional dominance (higher average contribution), and general dominance (weighted average across subset sizes).

The last dimension (the average contribution of each variable) provides a metric similar to semi-partial R2, and could be expressed as a proportion of the global R2 (relative contribution). The sum of these relative contributions is equal 100%.

`tbl_dominance()` is a quick helper to perform a dominance analysis with `dominanceanalysis::dominanceAnalysis()` and to display the average total contribution variable in a nicely formatted `gt::gt()` table. The

function also displays the relative contribution (i.e. the contribution of each variable expressed as a proportion of the R2).

To be noted, alternative implementations of dominance analysis in **R** include `parameters::dominance_analysis()` or `domir::domin()`.

Note

Regarding eta-squared indicators, more details are provided in a dedicated vignette of the `effectsize` package. This vignette also presents variations of these indicators. Some explanations are also available in the documentation of the `GAMLj` package for **Jamovi**.

To be noted, `GAMLj` highlights an potential issue regarding the computation of total variance in `effectsize::eta_squared(partial = FALSE)`. The `effectsize` package sum the values displayed in the Anova object instead of performing a null model. Here, we rely on `total_deviance()` and therefore estimates for eta-squared are not equal to those performed by `effectsize::eta_squared(partial = FALSE)`. To be noted, `effectsize::eta_squared(partial = TRUE)` (partial eta-squared), is not impacted by this difference in approaches.

Examples

```
# Linear model
m <- lm(Sepal.Length ~ Sepal.Width + Species + Petal.Length, data = iris)
m |> contributions()


m |> tbl_contributions()
m |> tbl_dominance()
m |> tbl_contributions(type = 1)

# GLM
m2 <- glm(Survived ~ ., data = titanic, family = binomial)
m2 |> contributions()
m2 |> tbl_contributions()
m2 |> tbl_contributions(decimals = 2)
m2 |> tbl_contributions(show = "Relative", notes = FALSE)
m2 |> tbl_dominance()

# custom column labels
tbl <- m2 |> tbl_contributions()
colnames(tbl$`_data`) # list of columns
tbl |>
  gt::cols_label(
    Predictor = "Variable",
    Total = "Custom label (total)",
    Partial = gt::html("Custom <em>label</em> with <strong>HTML</strong>"),
    Relative = gt::md("Custom _label_ with **markdown**")
  )

# interaction terms
m3 <- glm(
  Survived ~ Class * Sex + Age,
  data = titanic,
```

```
  family = binomial,
  contrasts = list(Class = contr.sum, Sex = contr.sum, Age = contr.sum)
)
m3 |> tbl_contributions(type = "III")

# Survey-weighted GLM
library(survey)
m4 <- svyglm(
  Survived ~ Class + Sex + Age,
  design = srvyr::as_survey(titanic),
  family = quasibinomial
)
m4 |> tbl_contributions(type = "drop1")
m4 |> tbl_dominance()
```

`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()
```

da.svyglm.fit *Fit indices for dominance analysis of survey-weighted GLM*

Description

Provides support of `survey::svyglm()` for `dominanceanalysis::dominanceAnalysis()`.

Usage

```
da.svyglm.fit(original.model, newdata = NULL, ...)
```

Arguments

`original.model`	Original fitted model
`newdata`	Data used in update statement
`...`	ignored

deprecated *Deprecated functions*

Description

Deprecated functions

Usage

```
plot_maihda_predictions_by(
  x,
  by = NULL,
  scale = c("response", "link"),
  which = c("null", "adjusted"),
  sort = TRUE
)
```

Arguments

`x`	a MAIHDA object (`maihda_analysis` or `maihda_model`); for `tbl_maihda()` it could also be a list of `maihda_model` objects; for `tbl_partially_adjusted_maihda()`, only a `maihda_analysis` computed with `MAIHDA::maihda(decomposition = "two-model")` is allowed; for `tbl_strata_info()`, the result of `MAIHDA::make_strata()` is also accepted
`by`	<tidy-select> list of variables to compare by

scale	scale for the predicted stratum values: "response" (default), "link", or "random_effect" (random effect only on the link scale); for a cumulative (ordinal) model the response scale is the expected category score.
which	For a two-model analysis, which model's predictions to rank the strata by: "null" (default) or "adjusted". Ignored for a crossed-dimensions analysis or a single model.
sort	should the plot be sorted?

---

grouped_tbl_pivot_wider *Helpers for grouped tables generated with* gtsummary

---

Description

A series of helpers for grouped tables generated by gtsummary::tbl_stack() or gtsummary::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 table generated with gtsummary::tbl_stack() or 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()

t1 <- gtsummary::trial |>
  gtsummary::tbl_summary(include = grade, by = trt)
t2 <- gtsummary::trial |>
  gtsummary::tbl_summary(include = stage, by = trt)

gtsummary::tbl_stack(list(t1, t2), group_header = c("Table 1", "Table 2")) |>
  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()`). `svyttest_oneway()` is designed to compare means between sub-groups for survey objects. It is based on `survey::svyttest()` for comparing 2 means, and on `svyoneway()` for comparing 3 means or more.

Usage

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

svyttest_oneway(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")



iris |>
  srvyr::as_survey() |>
  tbl_svysummary(
    include = Petal.Length,
    by = Species
   ) |>
   add_p(test = all_continuous() ~ svyttest_oneway)
```

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,
  missing_text = NULL,
  overall_string = NULL,
  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,
  missing_text = NULL,
  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.

`missing_text` (character)
String indicating text shown on missing row.

`overall_string` (character)
Optional string to name the *overall* column.

`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.

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_prop_n()` also modifies default method for `gtsummary::add_ci.tbl_summary()` (`"wilson"` for categorical variables, `"t.test"`, i.e. mean confidence interval, for continuous variables if `mean_sd = TRUE`, `"wilcox.test"`, i.e. confidence interval of the pseudomedian, for continuous variables if `mean_sd = FALSE`). Finally, `theme_gtsummary_prop_n()` also modifies default tests for `gtsummary::add_p.tbl_summary()` for continuous variables if `mean_sd = TRUE` (`"t.test"` for comparing 2 groups, or `"oneway.test"` for 3 groups or more). If `mean_sd = FALSE`, the default tests for continuous variables remain `"wilcox.test"` (2 groups) or `"kruskal.test"` (3 groups or more). For categorical variables, `"chisq.test.no.correct"` and `"fisher.test"` are used by default. See `theme_gtsummary_fisher_simulate_p()` to change the default test for categorical variables.

`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_unweighted_n()` also modifies default method for `gtsummary::add_ci.tbl_svysummary()` (`"svyprop.logit"` for categorical variables, `"svymean"`, i.e. mean confidence interval, for continuous variables if `mean_sd = TRUE`, `"svymedian.mean"`, i.e. confidence interval of the median, for continuous variables if `mean_sd = FALSE`). Finally, `theme_gtsummary_unweighted_n()` also modifies default tests for `gtsummary::add_p.tbl_sv` for continuous variables if `mean_sd = TRUE` (`svyttest_oneway` which calls `survey::svyttest()` for comparing 2 means and `svyoneway()` for comparing 3 means or more). If `mean_sd = FALSE`, the default tests for continuous variables remain `"svy.wilcox.test"` which used a designed-based Wilcoxon test (2 groups) or Kruskal-Wallis test (3 groups or more). For categorical variables, `"svy.chisq.test"`is used by default.

`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)

add_glance_header(x, header = "Summary statistics")
```

Arguments

x	A gtsummary object.
indent	An integer indicating how many space to indent text.
header	String of the header to place above the glance statistics

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)
```

---

long_to_seq *Transform a data frame from long format to a sequence object*

Description

Transform a data frame from long format to a sequence object

Usage

```
long_to_seq(
  data,
  id,
  time,
  outcome,
  alphabet = "auto",
  labels = "auto",
  cnames = "auto",
  cpal = "auto",
  missing.color = "#BBBBBB",
  ...
)
```

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
outcome	<tidy-select> Variable defining the status

alphabet	Optional vector containing the alphabet (the list of all possible states). If alphabet = "auto" will be automatically determined from outcome. If outcome is a labelled vector (haven_labelled class), it will be derived from the value labels (using the values). If outcome is a factor, the factor will be transformed to a numeric vector with as.integer() and the corresponding numeric values will be used as the alphabet. In all other cases, will be equal to NULL (see TraMineR::seqdef()).
labels	An optional vector containing state labels used for graphics. If labels = "auto" will be automatically determined from outcome. If outcome is a labelled vector (haven_labelled class), it will be derived from the value labels (using the labels). If outcome is a factor, the levels of the factor will be used. In all other cases, will be equal to NULL
cnames	An optional vector containing names of the different time points. If cnames = "auto", it will use the observed values from time.
cpal	An optional colour palette for representing the states in the graphics. If cpal = "auto", a palette will be generated with safe_pal().
missing.color	Alternative colour for representing missing values inside the sequences.
...	Additional arguments passed to TraMineR::seqdef()

Value

An object of class stslist.

See Also

TraMineR::seqdef()

Examples

```
library(TraMineR)

# generating a data frame in long format
data("biofam")
d <-
  biofam |>
  dplyr::mutate(id_ind = rownames(biofam)) |>
  dplyr::select(id_ind, dplyr::starts_with("a")) |>
  tidyr::pivot_longer(
    cols = dplyr::starts_with("a"),
    names_to = "age",
    names_prefix = "a",
    values_to = "life_state"
  ) |>
  dplyr::mutate(
    age = as.integer(age),
    life_state2 = dplyr::case_when(
      life_state == 0 ~ "P",
      life_state == 1 ~ "L",
```

```
      life_state == 2 ~ "M",
      life_state == 3 ~ "LM",
      life_state == 4 ~ "C",
      life_state == 5 ~ "LC",
      life_state == 6 ~ "LMC",
      life_state == 7 ~ "D"
    )
  ) |>
  labelled::set_value_labels(
    life_state = c(
      "Parent" = 0,
      "Left" = 1,
      "Married" = 2,
      "Left & Married" = 3,
      "Child" = 4,
      "Left & Child" = 5,
      "Left & Married & Child" = 6,
      "Divorced" = 7
    ),
    life_state2 = c(
      "Parent" = "P",
      "Left" = "L",
      "Married" = "M",
      "Left & Married" = "LM",
      "Child" = "C",
      "Left & Child" = "LC",
      "Left & Married & Child" = "LMC",
      "Divorced" = "D"
    )
  ) |>
  dplyr::mutate(
    life_state3 = labelled::to_factor(life_state),
    life_state4 = unclass(life_state2)
  )

d |> long_to_seq(id = id_ind, time = age, outcome = life_state) |> head(10)
d |> long_to_seq(id = id_ind, time = age, outcome = life_state2) |> head(10)
d |> long_to_seq(id = id_ind, time = age, outcome = life_state3) |> head(10)
d |> long_to_seq(id = id_ind, time = age, outcome = life_state4) |> head(10)
```

`mean_sd` *Compute means, standard deviations and confidence intervals by sub-groups*

Description

`mean_sd()` lets you quickly compute mean and standard deviation by sub-groups. Use `.conf.int = TRUE` to also return confidence intervals of the mean.

Usage

```
mean_sd(data, ...)

## S3 method for class 'data.frame'
mean_sd(
  data,
  ...,
  .by = NULL,
  .drop = FALSE,
  .drop_na_by = FALSE,
  .conf.int = FALSE,
  .conf.level = 0.95,
  .options = NULL
)

## S3 method for class 'survey.design'
mean_sd(
  data,
  ...,
  .by = NULL,
  .drop = FALSE,
  .drop_na_by = FALSE,
  .conf.int = FALSE,
  .conf.level = 0.95,
  .options = NULL
)

## Default S3 method:
mean_sd(
  data,
  ...,
  .drop = FALSE,
  .conf.int = FALSE,
  .conf.level = 0.95,
  .options = NULL
)
```

Arguments

data	A vector, a data frame, data frame extension (e.g. a tibble), or a survey design object.
...	<data-masking> Variable(s) for which to compute mean and standard deviation.
.by	<tidy-select> Optional additional variables to group by (in addition to those eventually previously declared using dplyr::group_by()).
.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::t.test() or srvyr::survey_mean().

Value

A tibble. Column "n" reports the number of valid observations and "missing" the number of missing (NA) observations, unweighted for survey objects.

A tibble with one row per group.

Examples

```
# using a vector
iris$Petal.Length |> mean_sd()

# one variable
iris |> mean_sd(Petal.Length)
iris |> mean_sd(Petal.Length, .conf.int = TRUE)
iris |> mean_sd(Petal.Length, .by = Species)
mtcars |> mean_sd(mpg, .by = c(cyl, gear))

# two variables
iris |> mean_sd(Petal.Length, Petal.Width)
iris |> mean_sd(dplyr::pick(dplyr::starts_with("Petal")), .by = Species)

# missing values
d <- iris
d$Petal.Length[1:10] <- NA
d |> mean_sd(Petal.Length)
d |> mean_sd(Petal.Length, .by = Species)


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

ds <- srvyr::as_survey(iris)
ds |> mean_sd(Petal.Length, .by = Species, .conf.int = TRUE)
```

---

median_iqr *Compute median, quartiles and interquartile range by sub-groups*

Description

median_iqr() lets you quickly compute median, quartiles and interquartile range by sub-groups. Use .outliers = TRUE to also return whiskers and outliers (see ggplot2::stat_boxplot()).

Usage

```
median_iqr(data, ...)

## S3 method for class 'data.frame'
median_iqr(
  data,
  ...,
  .by = NULL,
  .drop = FALSE,
  .drop_na_by = FALSE,
  .outliers = FALSE
)

## S3 method for class 'survey.design'
median_iqr(
  data,
  ...,
  .by = NULL,
  .drop = FALSE,
  .drop_na_by = FALSE,
  .outliers = FALSE
)

## Default S3 method:
median_iqr(data, ..., .drop = FALSE, .outliers = FALSE)
```

Arguments

`data`	A vector, a data frame, data frame extension (e.g. a tibble), or a survey design object.
`...`	<data-masking> Variable(s) for which to compute median, quartiles and interquartile range.
`.by`	<tidy-select> Optional additional variables to group by (in addition to those eventually previously declared using `dplyr::group_by()`).
`.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()`).
`.outliers`	If `TRUE`, will estimate whiskers and outliers.

Value

A tibble. Column `"n"` reports the number of valid observations and `"missing"` the number of missing (`NA`) observations, unweighted for survey objects.

A tibble with one row per group.

Examples

```
# using a vector
iris$Petal.Length |> median_iqr()

# one variable
iris |> median_iqr(Petal.Length)
iris |> median_iqr(Petal.Length, .outliers = TRUE)
iris |> median_iqr(Petal.Length, .by = Species)
mtcars |> median_iqr(mpg, .by = c(cyl, gear))

# two variables
iris |> median_iqr(Petal.Length, Petal.Width)
iris |> median_iqr(dplyr::pick(dplyr::starts_with("Petal")), .by = Species)

# missing values
d <- iris
d$Petal.Length[1:10] <- NA
d |> median_iqr(Petal.Length)
d |> median_iqr(Petal.Length, .by = Species)


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

ds <- srvyr::as_survey(iris)
ds |> median_iqr(Petal.Length, .by = Species, .outliers = TRUE)
```

---

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_categorical *Plot a categorical variable by sub-groups*

Description

Plot one or several categorical variables by sub-groups. See proportion() for more details on the way proportions and confidence intervals are computed. Return a bar plot (see examples).

Usage

```
plot_categorical(
  data,
  outcome,
  na.rm = TRUE,
  by = NULL,
  stratified_by = NULL,
  drop_na_by = FALSE,
  convert_continuous = TRUE,
  ...,
  show_overall = TRUE,
  overall_label = "Overall",
  show_pvalues = TRUE,
  pvalues_test = c("fisher", "chisq"),
  pvalues_labeller = scales::label_pvalue(add_p = TRUE),
  pvalues_size = 3.5,
  pvalues_y = ifelse(flip, 1.05, 1),
  show_labels = TRUE,
  labels_labeller = scales::label_percent(1),
```

```
  labels_size = 3.5,
  labels_color = "auto",
  facet_labeller = ggplot2::label_wrap_gen(width = 50, multi_line = TRUE),
  flip = FALSE,
  minimal = FALSE,
  return_data = FALSE
)
```

Arguments

data	A data frame, data frame extension (e.g. a tibble), or a survey design object.
outcome	<tidy-select> List of categorical variables to be plotted.
na.rm	Should NA values be removed from the outcome?
by	<tidy-select> List of variables to group by (comparison is done separately for each variable).
stratified_by	<tidy-select> Variable to stratify by (only one outcome variable is accepted if a stratification is requested).
drop_na_by	Remove NA values in by variables?
convert_continuous	Should continuous by variables (with 5 unique values or more) be converted to quartiles (using cut_quartiles())?
...	Additional arguments passed to ggplot2::geom_bar().
show_overall	Display "Overall" column?
overall_label	Label for the overall column.
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.
pvalues_y	Y position of p-values.
show_labels	Display proportion labels?
labels_labeller	Labeller function for labels.
labels_size	Size of labels.
labels_color	Color of labels.
facet_labeller	Labeller function for strip labels.
flip	Flip x and y axis?
minimal	Should a minimal theme be applied? (no y-axis, no grid)
return_data	Return computed data instead of the plot?

Examples

```
titanic |>
  plot_categorical(
    Class,
    by = c(Age, Sex)
  )


titanic |>
  plot_categorical(
    Class,
    by = c(Age, Sex),
    show_overall = FALSE,
    flip = TRUE
  )

titanic |>
  plot_categorical(
    Class,
    by = c(Age, Sex),
    flip = TRUE,
    minimal = TRUE
  )
titanic |>
  plot_categorical(
    Age,
    by = Class,
    stratified_by = Sex
  )



gtsummary::trial |>
  plot_categorical(grade, by = c(age, stage, trt))
gtsummary::trial |>
  plot_categorical(grade, by = c(age, stage, trt), drop_na_by = TRUE)
gtsummary::trial |>
  plot_categorical(c(grade, stage), by = c(trt, response))
```

`plot_continuous` *Plot a continuous variable by sub-groups*

Description

Plot one or several continuous variables by sub-groups. See `median_iqr()` for more details on the way statistics are computed. Return a box plot (see examples).

Usage

```
plot_continuous(
  data,
  outcome,
  by = NULL,
  stratified_by = NULL,
  drop_na_by = FALSE,
  convert_continuous = TRUE,
  ...,
  show_overall = TRUE,
  overall_label = "Overall",
  show_pvalues = TRUE,
  pvalues_labeller = scales::label_pvalue(add_p = TRUE),
  pvalues_size = 3.5,
  facet_labeller = ggplot2::label_wrap_gen(width = 50, multi_line = TRUE),
  flip = FALSE,
  minimal = FALSE,
  free_scale = FALSE,
  return_data = FALSE
)
```

Arguments

data	A data frame, data frame extension (e.g. a tibble), or a survey design object.
outcome	<tidy-select> List of continuous variables to be plotted.
by	<tidy-select> List of variables to group by (comparison is done separately for each variable).
stratified_by	<tidy-select> Variable to stratify by (only one outcome variable is accepted if a stratification is requested).
drop_na_by	Remove NA values in by variables?
convert_continuous	Should continuous by variables (with 5 unique values or more) be converted to quartiles (using cut_quartiles())?
...	Additional arguments passed to ggplot2::geom_boxplot().
show_overall	Display "Overall" column?
overall_label	Label for the overall column.
show_pvalues	Display p-values in the top-left corner? p-values are computed with stats::kruskal.test() for data frames, and with survey::svyranktest() for survey objects.
pvalues_labeller	Labeller function for p-values.
pvalues_size	Text size for p-values.
facet_labeller	Labeller function for strip labels.
flip	Flip x and y axis?

minimal	Should a minimal theme be applied? (no y-axis, no grid)
free_scale	Allow y axis to vary between conditions?
return_data	Return computed data instead of the plot?

Examples

```
iris |>
  plot_continuous(Petal.Length, by = Species)

iris |>
  plot_continuous(
    dplyr::starts_with("Petal"),
    by = Species,
    free_scale = TRUE,
    fill = "lightblue",
    outlier.color = "red"
  )

iris |>
  plot_continuous(
    Petal.Length,
    by = Petal.Width,
    stratified_by = Species
  )



mtcars |>
  plot_continuous(
    mpg,
    by = c(cyl, gear),
    flip = TRUE,
    mapping = ggplot2::aes(fill = by)
  )

mtcars |>
  plot_continuous(
    mpg,
    by = cyl,
    stratified_by = gear
  )

# works with continuous by variables
mtcars |>
  plot_continuous(
    mpg,
    by = c(disp, drat),
    flip = TRUE,
    minimal = TRUE
  )

# works with survey object
```

```
iris |>
  srvyr::as_survey() |>
  plot_continuous(
    Petal.Length,
    by = c(Species, Petal.Width),
    flip = TRUE
  )
```

`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_means *Plot means by sub-groups*

Description

Plot one or several means by sub-groups. See `mean_sd()` for more details on the way means and confidence intervals are computed. By default, return a point plot, but other geometries could be used (see examples).

Usage

```
plot_means(
  data,
  outcome,
  by = NULL,
  stratified_by = NULL,
  drop_na_by = FALSE,
  convert_continuous = TRUE,
  geom = "point",
  ...,
  show_overall = TRUE,
  overall_label = "Overall",
  show_ci = TRUE,
  conf_level = 0.95,
  ci_color = "black",
  show_pvalues = TRUE,
  pvalues_labeller = scales::label_pvalue(add_p = TRUE),
  pvalues_size = 3.5,
  show_labels = TRUE,
  label_y = NULL,
  labels_labeller = scales::label_number(0.1),
  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,
  minimal = FALSE,
  free_scale = FALSE,
  return_data = FALSE
)
```

Arguments

data	A data frame, data frame extension (e.g. a tibble), or a survey design object.

outcome	<tidy-select> List of continuous variables to be plotted.
by	<tidy-select> List of variables to group by (comparison is done separately for each variable).
stratified_by	<tidy-select> Variable to stratify by (only one outcome variable is accepted if a stratification is requested).
drop_na_by	Remove NA values in by variables?
convert_continuous	Should continuous by variables (with 5 unique values or more) be converted to quartiles (using cut_quartiles())?
geom	Geometry to use for plotting means ("point" 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? p-values are computed with stats::oneway.test() for data frames, and with survey::svyttest() (2 groups) or svyoneway() (3 groups or more) for survey objects.
pvalues_labeller	Labeller function for p-values.
pvalues_size	Text size for p-values.
show_labels	Display mean labels?
label_y	Y position of labels. If NULL, will be auto-determined.
labels_labeller	Labeller function for labels.
labels_size	Size of labels.
labels_color	Color of 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?
minimal	Should a minimal theme be applied? (no y-axis, no grid)
free_scale	Allow y axis to vary between conditions?
return_data	Return computed data instead of the plot?

Examples

```
iris |>
  plot_means(Petal.Length, by = Species)

iris |>
  plot_means(Petal.Length, by = Petal.Width, stratified_by = Species)

iris |>
  plot_means(
    dplyr::starts_with("Petal"),
    by = Species,
    geom = "bar",
    fill = "lightblue",
    show_overall_line = TRUE
  )



mtcars |>
  plot_means(
    mpg,
    by = c(cyl, gear),
    size = 3,
    colour = "plum",
    flip = TRUE
  )

# works with continuous by variables
mtcars |>
  plot_means(
    mpg,
    by = c(disp, drat),
    fill = "plum",
    geom = "bar",
    flip = TRUE,
    minimal = TRUE
  )

# works with survey object
iris |>
  srvyr::as_survey() |>
  plot_means(
    Petal.Length,
    by = c(Species, Petal.Width),
    label_y = -1,
    size = 3,
    mapping = ggplot2::aes(colour = by),
    flip = TRUE
  )
```

`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,
  drop_na_by = 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,
  drop_na_by = 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?
drop_na_by	If TRUE, will remove any NA values observed in the by variables
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,
  label_y = NULL,
  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,
  minimal = 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 by 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?
label_y	Y position of labels. If NULL, will be auto-determined.
labels_labeller	Labeller function for labels.
labels_size	Size of labels.
labels_color	Color of 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?
minimal	Should a minimal theme be applied? (no y-axis, no grid)
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),
    fill = "lightblue",
    minimal = 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 = by),
    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, a data frame extension (e.g. a tibble), or a survey design object.
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.

For survey design objects, weights are not taken into account. Each individual trajectory as the same height.

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))
```

---

safe_pal *A safe discrete colour palette*

Description

Provides a safe colour palette for categorical variable. It is based on Paul Tol's colour schemes designed to be distinct for all people, including colour-blind readers, distinct from black and white, distinct on screen and paper, and matching well together. It is primarily based on the *bright* colour scheme implemented in khroma::scale_fill_bright(). This colour scheme include 7 colours, including a grey reserved for NA values. Therefore, scale_fill_safe() use the *bright* scheme only if 6 or less colours are needed (keeping the grey for any NA value). If 7 to 9 colours are needed, the *muted* scheme (cf. khroma::scale_fill_muted()) is used instead. Finally, if 10 or more colours are requested, the *rainbow* scheme is used (cf. khroma::scale_fill_discreterainbow()). This is a sequential colour scheme. Here, colour are randomly reordered to provide more contrasts between modalities.

Usage

```
safe_pal(reverse = FALSE)

scale_fill_safe(
  name = ggplot2::waiver(),
  ...,
  reverse = FALSE,
  aesthetics = "fill",
  na.value = "#BBBBBB"
)

scale_colour_safe(
  name = ggplot2::waiver(),
  ...,
  reverse = FALSE,
  aesthetics = "colour",
  na.value = "#BBBBBB"
)

scale_color_safe(
  name = ggplot2::waiver(),
  ...,
  reverse = FALSE,
  aesthetics = "colour",
  na.value = "#BBBBBB"
)
```

Arguments

reverse	A logical scalar: should the resulting vector of colours be reversed?
name	The name of the scale. Used as the axis or legend title. If ggplot2::waiver(), the default, the name of the scale is taken from the first mapping used for that aesthetic. If NULL, the legend title will be omitted.
...	Other arguments passed on to discrete_scale() to control name, limits, breaks, labels and so forth.
aesthetics	Character string or vector of character strings listing the name(s) of the aesthetic(s) that this scale works with. This can be useful, for example, to apply colour settings to the colour and fill aesthetics at the same time, via aesthetics = c("colour", "fill").
na.value	Colour to be used for NA values (if any).

Value

A palette function.

Examples

```
scales::show_col(safe_pal()(6))
scales::show_col(safe_pal(reverse = TRUE)(6))
scales::show_col(safe_pal()(9))
scales::show_col(safe_pal()(16))

ggplot2::ggplot(titanic) +
  ggplot2::aes(x = Age, fill = Class) +
  ggplot2::geom_bar() +
  scale_fill_safe()

ggplot2::ggplot(iris) +
  ggplot2::aes(x = Petal.Length, y = Petal.Width, colour = Species) +
  ggplot2::geom_point(size = 3) +
  scale_colour_safe()
```

`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
```

---

svyoneway *Test for Equal Means for survey design object*

Description

This function allows to compare several means using `survey::svyglm()`. More precisely, this is a wrapper for `survey::regTermTest(m, "group")` where `m <- survey::svyglm(x ~ group, design)`.

Usage

```
svyoneway(formula, design, ...)
```

Arguments

`formula`	a formula of the form `lhs ~ rhs` where `lhs` gives the sample values and `rhs` the corresponding groups
`design`	a survey design object
`...`	additional parameters passed to `survey::regTermTest()`

Value

an object of class `"htest"`

See Also

`stats::oneway.test()` for classic data frames

Examples

```
svyoneway(
  Petal.Length ~ Species,
  design = srvyr::as_survey(iris)
)
```

---

tbl_maihda *Table summary of for MAIHDA analysis*

Description

[Experimental] Helpers to generate formatted tables of a MAIHDA analysis as proposed by Evans et al. (*SSM - Population Health* 2024, doi:10.1016/j.ssmph.2024.101664). It relies on the MAIHDA package. This package being under active development, the proposed functions here are experimental.

Usage

```
tbl_maihda(
  x,
  conf.level = 0.95,
  exponentiate = FALSE,
  ...,
  global_p = FALSE,
  bootstrap_vpc = FALSE,
  bootstrap_pcv = FALSE,
  n_boot = 1000,
  twomodels_labels = c("Null model", "Adjusted model"),
  statistics_header = "Summary statistics",
  statistics_labels = list(bsv = "Between-stratum variance", bssd =
    "Between-stratum standard deviation", vpc =
    "Variance Partition Coefficient (VPC / adjusted ICC)", pcv =
    "Proportional Change in Variance (PCV)", auc =
    "Area Under Receiver Operating Characteristic Curve (AUC)", mor =
    "Median Odds Ratio (MOR)", cs =
    "Context share (between-context component of unexplained variance)", as =
    "Additive share of between-strata variance", is =
    "Interaction share of between-strata variance", r2cond =
    "Conditional Nakagawa's R2 (fixed + random effects)",
     r2marg =
    "Marginal Nakagawa's R2 (fixed effects only)", uicc =
    "Unadjusted ICC (intraclass correlation coefficient)", avar =
    "Additive (sum of dimension main effects) variance", ivar =
   "Intersectional interaction variance", tvar = "Total between-strata variance", decomp
    = "Additive vs. Intersectional Decomposition (crossed-dimensions)", pdav =
    "Per-dimension additive variance"),
  statistics_include = -dplyr::any_of("bssd"),
  notes = TRUE,
 notes_labels = list(n_strata = "Strata:", nobs = "Observations:", engine = "Engine:",
    family = "Family:", context = "Variable(s) in context:"),
  hide_coefficients = FALSE,
  return_data = FALSE
)

tbl_partially_adjusted_maihda(
  x,
  conf.level = 0.95,
  exponentiate = FALSE,
  ...,
  global_p = FALSE,
  bootstrap_vpc = FALSE,
  bootstrap_pcv = FALSE,
  n_boot = 1000,
  twomodels_labels = c("Null model", "Fully adjusted model"),
  statistics_header = "Summary statistics",
```

```
  statistics_labels = list(bsv = "Between-stratum variance", bssd =
    "Between-stratum standard deviation", vpc =
    "Variance Partition Coefficient (VPC / adjusted ICC)", pcv =
    "Proportional Change in Variance (PCV)", auc =
    "Area Under Receiver Operating Characteristic Curve (AUC)", mor =
    "Median Odds Ratio (MOR)", cs =
    "Context share (between-context component of unexplained variance)", as =
    "Additive share of between-strata variance", is =
    "Interaction share of between-strata variance", r2cond =
    "Conditional Nakagawa's R2 (fixed + random effects)",
     r2marg =
    "Marginal Nakagawa's R2 (fixed effects only)", uicc =
    "Unadjusted ICC (intraclass correlation coefficient)"),
  statistics_include = -dplyr::any_of("bssd"),
  notes = TRUE,
 notes_labels = list(n_strata = "Strata:", nobs = "Observations:", engine = "Engine:",
    family = "Family:", context = "Variable(s) in context:"),
  hide_coefficients = FALSE,
  return_data = FALSE
)

calculate_partially_adjusted_maihda(
  x,
  conf.level = 0.95,
  ...,
  global_p = FALSE,
  bootstrap_vpc = FALSE,
  bootstrap_pcv = FALSE,
  n_boot = 1000,
  twomodels_labels = c("Null model", "Fully adjusted model")
)

tbl_strata_info(
  x,
  breaks = c(10, 20, 30, 50, 100),
  type = c("nested", "exclusive"),
 column_labels = list(size = "Sample size per stratum", n = "Number of strata", prop =
    "Proportion of strata"),
  total_label = "Total number of strata:"
)

tbl_strata_predictions(
  x,
  n_strata = Inf,
  scale = c("response", "link", "random_effect"),
  which = c("null", "adjusted"),
 column_labels = list(rank = "Rank", n = "n", predicted = "Predicted", ci = "95% CI"),
  group_labels = list("highest", "lowest"),
```

```
  digits = 1L
)

get_strata_predictions(
  x,
  n_strata = Inf,
  scale = c("response", "link", "random_effect"),
  which = c("null", "adjusted"),
  group_labels = list("highest", "lowest")
)

plot_strata_predictions(
  x,
  by = NULL,
  geom = c("point", "bar"),
  n_strata = Inf,
  scale = c("response", "link", "random_effect"),
  which = c("null", "adjusted"),
  sort = TRUE,
  highlight_n_below = NULL,
  show_mean_line = FALSE,
  mean_line_color = "#AA4499",
  mean_line_type = "dashed",
  mean_line_width = 0.5
)

glance_maihda_model(x, bootstrap_vpc = FALSE, conf.level = 0.95, n_boot = 1000)
```

Arguments

x	a MAIHDA object (maihda_analysis or maihda_model); for tbl_maihda() it could also be a list of maihda_model objects; for tbl_partially_adjusted_maihda(), only a maihda_analysis computed with MAIHDA::maihda(decomposition = "two-model") is allowed; for tbl_strata_info(), the result of MAIHDA::make_strata() is also accepted
conf.level	confidence level for confidence/credible intervals
exponentiate	should model coefficients be exponentiated?
...	additional parameters passed to gtsummary::tbl_regression()
global_p	display global p-value instead of terms p-value (see gtsummary::add_global_p()), not available if engine = "wemix".
bootstrap_vpc	logical indicating whether to compute parametric bootstrap confidence intervals for VPC/ICC; supported only for engine = "lme4"; when using engine = "brms", posterior credible intervals are always returned; could be very time-consuming; cf. MAIHDA::summary.maihda_model() for more details.
bootstrap_pcv	logical indicating whether to compute bootstrap confidence intervals for the PCV; implemented only for engine = "lme4" and if x is an maihda_analysis

object, otherwise PCV should be manually computed and added to models before calling `tbl_maihda()` (see examples); cf. `MAIHDA::calculate_pcv()` for more details.

`n_boot` number of bootstrap samples when bootstrap is used to estimate confidence intervals

`twomodels_labels`
for a two-model MAIHDA analysis, labels for the two models

`statistics_header`
string header of the summary statistics

`statistics_labels`
name list of labels for the summary statistics

`statistics_include`
<tidy-select>
names of summary statistics to be included: must be column names of the tibble returned by `glance_maihda_model()`

`notes` display some notes (number of strata, of observations, engine, model family) about the analysis?

`notes_labels` name list of labels for the notes

`hide_coefficients`
should model coefficients be hidden? (display only summary statistics)

`return_data` return a data frame instead of a table

`breaks` breaks for sample size per stratum

`type` type of table (nested or exclusive size categories)

`column_labels` named list of column labels

`total_label` string of the total label in the notes

`n_strata` number of strata to show at each end (top and bottom), use `Inf` or `NULL` to show all strata

`scale` scale for the predicted stratum values: "response" (default), "link", or "random_effect" (random effect only on the link scale); for a cumulative (ordinal) model the response scale is the expected category score.

`which` For a two-model analysis, which model's predictions to rank the strata by: `"null"` (default) or `"adjusted"`. Ignored for a crossed-dimensions analysis or a single model.

`group_labels` labels for group names

`digits` number of decimals for predictions

`by` <tidy-select>
list of variables to compare by

`geom` geometry to use for plotting proportions ("point" by default).

`sort` should the plot be sorted?

`highlight_n_below`
highlight strata with a number of observations below this number (`NULL` for not highlight, incompatible with `geom = "bar"`)

show_mean_line add a vertical line displaying the mean prediction?

mean_line_color
color of the mean line

mean_line_type type of the mean line

mean_line_width
width of the mean line

Details

tbl_maihda() is intended to replicate Table 3 of Evans et al. 2024, with fixed effects, between-stratum variance and model summary statistics including VPC (variance partition coefficient) and PCV (proportional change in variance). It accepts a maihda_analysis object created with MAIHDA::maihda(), a single maihda_model created with MAIHDA::fit_maihda() or a list of several maihfda_model objects. For this last case, PCV should be manually added to the models to be displayed (see examples). When bootstrapped confidence intervals are requested, the computation time could be very long. In such case, you can call tbl_maihda() with return_data = TRUE, save the result and pass it to tbl_maihda() to display the table.

tbl_partially_adjusted_maihda() is an helper allowing to compute and display all partially adjusted models (see examples). When bootstrapped confidence intervals are requested, the computation could be very long. In such case, it could be relevant to use calculate_partially_adjusted_maihda() to save the result of the computation and then to pass it directly to tbl_maihda().

tbl_strata_info() is intended to replicate Table 2, showing the number of strata having a certain sample size.

tbl_strata_predictions() is intended to replicate Table 4, showing the strata with the highest and the lowest predicted value. If a maihda_analysis object is passed to tbl_strata_predictions(), the null model is taken into account by default for computing the predicted values, following the behavior of MAIHDA::maihda_table(). It should be noted that in Evans et al. 2024, the authors used the adjusted model, which could be done with the argument which = "adjusted".

plot_strata_predictions() allows to visually compare predicted values by strata according to one or several specific variable defining the strata.

To be noted, themes from the gtsummary package are taken into account for formatting the different values.

Examples

```
theme_gtsummary_bold_labels()

# gaussian model

data("maihda_health_data", package = "MAIHDA")
a <- MAIHDA::maihda(
  BMI ~ Age + Gender + Race + (1 | Gender:Race),
  data = maihda_health_data
)

a |> tbl_strata_info(breaks = c(50, 100, 150))
a |> tbl_maihda()
```

```
a |> tbl_strata_predictions()
a |> plot_strata_predictions()
a |> plot_strata_predictions(by = Race)

# a binomial example

titanic |>
  MAIHDA::make_strata(c("Age", "Class")) |>
  tbl_strata_info()

m <- MAIHDA::maihda(
  Survived ~ Age + Sex + Class + (1 | Age:Sex:Class),
  data = titanic,
  family = binomial
)

m |> tbl_strata_info()
m |> tbl_strata_info(type = "exclusive")
m |> tbl_maihda(exponentiate = TRUE)
m |>
  tbl_maihda(
    statistics_include = dplyr::any_of(c("vpc", "pcv")),
    notes = FALSE
  )
m |> tbl_strata_predictions()
m |> tbl_strata_predictions(which = "adjusted", n_strata = 3)
m |> plot_strata_predictions()
m |> plot_strata_predictions(geom = "bar")
m |> plot_strata_predictions(n_strata = 3L)
m |> plot_strata_predictions(by = Sex, show_mean_line = TRUE)
m |> plot_strata_predictions(by = c(Sex, Age))
m |> plot_strata_predictions(highlight_n_below = 20)
m |> plot_strata_predictions(by = Age, highlight_n_below = 20)
m |> plot_strata_predictions(scale = "random_effect", which = "adjusted")
m |>
  plot_strata_predictions(by = Sex) +
  ggplot2::facet_grid(
    rows = ggplot2::vars(Age),
    scales = "free_y",
    space = "free_y"
  )

# Partially adjusted models

m0 <- MAIHDA::fit_maihda(
  Survived ~ 1 + (1 | Age:Sex:Class),
  data = titanic,
  family = binomial
)
m1 <- MAIHDA::fit_maihda(
  Survived ~ Age + (1 | Age:Sex:Class),
  data = titanic,
  family = binomial
```

```
)
m2 <- MAIHDA::fit_maihda(
  Survived ~ Sex + (1 | Age:Sex:Class),
  data = titanic,
  family = binomial
)
m3 <- MAIHDA::fit_maihda(
  Survived ~ Class + (1 | Age:Sex:Class),
  data = titanic,
  family = binomial
)

# manually adding PCV
m1$pcv <- MAIHDA::calculate_pcv(m0, m1)
m2$pcv <- MAIHDA::calculate_pcv(m0, m2)
m3$pcv <- MAIHDA::calculate_pcv(m0, m3)

list(Null = m0, Age = m1, Sex = m2, Class = m3) |>
  tbl_maihda(exponentiate = TRUE, global_p = TRUE)

# in one call
m |> tbl_partially_adjusted_maihda(exponentiate = TRUE)

# crossed-dimension MAIHDA
cdm <- MAIHDA::maihda(
  Survived ~ 1 + (1 | Age:Sex:Class),
  data = titanic,
  family = binomial,
  decomposition = "crossed-dimensions"
)
cdm |> tbl_maihda(exponentiate = TRUE)
cdm |> tbl_maihda(hide_coefficients = TRUE)

cdm2 <- MAIHDA::maihda(
  BMI ~ Gender + (1 | Education:Race),
  data = maihda_health_data,
  decomposition = "crossed-dimensions",
  context = "Age"
)

# sample-weighted data
if (rlang::is_installed("WeMix")) {

d <- titanic
d$weight <- runif(nrow(d), min = .75, max = 1.25)
wm <- MAIHDA::maihda(
  Survived ~ Age + Sex + Class + (1 | Age:Sex:Class),
  data = d,
  family = binomial,
  sampling_weights = "weight",
  engine = "wemix"
)
wm |> tbl_maihda(exponentiate = TRUE)
```

```
}
```

`titanic` *Titanic data set in long format*

Description

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

Usage

```
titanic
```

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

∗ **datasets**
titanic, 64
∗ **hplot**
plot_categorical, 29
plot_continuous, 31
plot_means, 35
plot_multiple_answers, 38
plot_proportions, 41
plot_trajectories, 46
safe_pal, 52
∗ **htest**
gtsummary_test, 13
svyoneway, 55
∗ **logic**
is_different, 18
∗ **manip**
combine_answers, 3
cut_quartiles, 9
long_to_periods, 20
long_to_seq, 21
periods_to_long, 28
unrowwise, 64
∗ **models**
add_interactions_by_step, 3
contributions, 4
da.svyglm.fit, 10
grouped_tbl_pivot_wider, 11
observed_vs_theoretical, 27
step_with_na, 54
tbl_maihda, 56
∗ **tree**
plot_inertia_from_tree, 34
∗ **univar**
mean_sd, 23
median_iqr, 25
proportion, 49
round_preserve_sum, 51
∗ **utilities**
gtsummary_themes, 14
gtsummary_utilities, 16
install_dependencies, 17
leading_zeros, 19
view_dictionary, 65

add_glance_header *(*gtsummary_utilities*)*, 16
add_interactions_by_step, 3
as.integer(), *22*

base::cut(), *9*
base::formatC(), *19*
base::round(), *51*
base::sprintf(), *19*
bold_variable_group_headers *(*gtsummary_utilities*)*, 16

calculate_partially_adjusted_maihda *(*tbl_maihda*)*, 56
car::Anova(), *5*, *7*
combine_answers, 3
combine_answers(), *38*
contributions, 4
cumdifferent *(*is_different*)*, 18
cut_quartiles, 9

da.svyglm.fit, 10
datasets::Titanic, *64*
deprecated, 10
dominanceanalysis::dominanceAnalysis(), *4*, *5*, *7*, *10*
dplyr::count(), *49*
dplyr::group_by(), *24–26*, *50*, *64*
dplyr::rowwise(), *64*
dplyr::tibble(), *42*
DT::datatable(), *65*
dummy_proportions *(*plot_proportions*)*, 41

FactoMineR::HCPC, *34*
fisher.simulate.p *(*gtsummary_test*)*, 13

get_inertia_from_tree
(plot_inertia_from_tree), 34
get_strata_predictions (tbl_maihda), 56
ggplot2::geom_bar(), *30*
ggplot2::geom_boxplot(), *32*
ggplot2::geom_tile(), *48*
ggplot2::stat_boxplot(), *25*
ggplot2::waiver(), *53*
ggupset, *38*
glance_maihda_model (tbl_maihda), 56
grouped_tbl_pivot_wider, 11
gt, *4*
gt::gt(), *7*
gtsummary, *16*, *61*
gtsummary::add_ci.tbl_summary(), *15*
gtsummary::add_ci.tbl_svysummary(), *15*
gtsummary::add_global_p(), *11, 12*, *59*
gtsummary::add_p.tbl_summary(), *15*
gtsummary::add_p.tbl_svysummary(), *15*
gtsummary::as_gt(), *11*
gtsummary::bold_labels(), *11*, *15*
gtsummary::modify_bold(), *17*
gtsummary::modify_indent(), *17*
gtsummary::modify_italic(), *17*
gtsummary::tbl_regression(), *11, 12*, *59*
gtsummary::tbl_stack(), *11, 12*
gtsummary::tbl_summary(), *14*
gtsummary::tbl_svysummary(), *15*
gtsummary::tests, *13*
gtsummary::theme_gtsummary_mean_sd(),
15
gtsummary_test, 13
gtsummary_themes, 14
gtsummary_utilities, 16

indent_labels (gtsummary_utilities), 16
indent_levels (gtsummary_utilities), 16
install_dependencies, 17
is_different, 18
is_equal (is_different), 18
italicize_variable_group_headers
(gtsummary_utilities), 16

khroma::scale_fill_bright(), *52*
khroma::scale_fill_discreterainbow(),
52
khroma::scale_fill_muted(), *52*

labelled::look_for(), *65*
leading_zeros, 19
long_to_periods, 20
long_to_periods(), *29*, *48*
long_to_seq, 21

MAIHDA, *56*
MAIHDA::calculate_pcv(), *60*
MAIHDA::fit_maihda(), *61*
MAIHDA::maihda(), *61*
MAIHDA::maihda_table(), *61*
MAIHDA::make_strata(), *10*, *59*
MAIHDA::summary.maihda_model(), *59*
mean_sd, 23
mean_sd(), *35*
median_iqr, 25
median_iqr(), *31*
multinom_add_global_p_pivot_wider
(grouped_tbl_pivot_wider), 11

nnet::multinom(), *55*
num_cycle (is_different), 18

observed_vs_theoretical, 27

pak::pkg_install(), *17*
periods_to_long, 28
periods_to_long(), *20*, *48*
plot_categorical, 29
plot_continuous, 31
plot_inertia_from_tree, 34
plot_maihda_predictions_by
(deprecated), 10
plot_means, 35
plot_multiple_answers, 38
plot_multiple_answers_dodge
(plot_multiple_answers), 38
plot_periods (plot_trajectories), 46
plot_proportions, 41
plot_strata_predictions (tbl_maihda), 56
plot_trajectories, 46
proportion, 49
proportion(), *29*, *38*, *41*

renv::dependencies(), *17, 18*
round_preserve_sum, 51

safe_pal, 52
safe_pal(), *22*
scale_color_safe (safe_pal), 52
scale_colour_safe (safe_pal), 52

scale_fill_safe *(safe_pal)*, 52
srvyr::survey_mean(), *25*
srvyr::survey_prop(), *50*
stats::add1(), *5*, *7*
stats::anova(), *5*, *6*
stats::as.hclust(), *34*
stats::chisq.test(), *30*, *43*
stats::deviance(), *6*
stats::drop1(), *5*, *7*
stats::fisher.test(), *13*, *30*, *43*
stats::glm(), *27*, *55*
stats::hclust, *34*
stats::kruskal.test(), *32*
stats::lm(), *27*, *55*
stats::oneway.test(), *36*, *56*
stats::prop.test(), *50*
stats::step(), *3*, *54*
stats::t.test(), *25*
step_with_na, 54
stratified_by *(plot_proportions)*, 41
style_grouped_tbl *(grouped_tbl_pivot_wider)*, 11
survey::regTermTest(), *56*
survey::svychisq(), *30*, *43*
survey::svyglm(), *7*, *10*, *54–56*
survey::svyranktest(), *32*
survey::svyttest(), *13*, *15*, *36*
survival::coxph(), *55*
svyoneway, 55
svyoneway(), *13*, *15*, *36*
svyttest_oneway *(gtsummary_test)*, 13

tbl_contributions *(contributions)*, 4
tbl_dominance *(contributions)*, 4
tbl_maihda, 56
tbl_partially_adjusted_maihda *(tbl_maihda)*, 56
tbl_strata_info *(tbl_maihda)*, 56
tbl_strata_predictions *(tbl_maihda)*, 56
theme_gtsummary_bold_labels *(gtsummary_themes)*, 14
theme_gtsummary_fisher_simulate_p *(gtsummary_themes)*, 14
theme_gtsummary_prop_n *(gtsummary_themes)*, 14
theme_gtsummary_unweighted_n *(gtsummary_themes)*, 14
tidyr::pivot_longer(), *48*
titanic, 64
to_DT *(view_dictionary)*, 65
total_deviance *(contributions)*, 4
TraMineR::seqdef(), *22*

unrowwise, 64
utils::old.packages(), *18*

view_detailed_dictionary *(view_dictionary)*, 65
view_dictionary, 65