Package ‘ggstats’

September 28, 2023

Type Package

Title Extension to 'ggplot2' for Plotting Stats

Version 0.5.0

Description Provides new statistics, new geometries and new positions for 'ggplot2' and a suite of functions to facilitate the creation of statistical plots.

License GPL (>= 3)


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

Imports broom.helpers (>= 1.14.0), cli, dplyr, forcats, ggplot2 (>= 3.4.0), lifecycle, magrittr, patchwork, purrr, rlang, scales, stats, stringr, tidyr

Suggests betareg, broom, emmeans, glue, knitr, labelled (>= 2.11.0), reshape, rmarkdown, nnet, parameters, pscl, testthat (>= 3.0.0), spelling, survey, survival, vdiff

Encoding UTF-8

RoxygenNote 7.2.3

Config/testthat/edition 3

Language en-US

VignetteBuilder knitr

NeedsCompilation no

Author Joseph Larmarange [aut, cre] (https://orcid.org/0000-0001-7970-000X)

Maintainer Joseph Larmarange <joseph@larmarange.net>

Repository CRAN

Date/Publication 2023-09-28 12:40:02 UTC
R topics documented:

*augment_chisq_add_phi*                  2
*geom_stripped_rows*                     3
*ggcoef_model*                          5
*gglikert*                              15
*ggsurvey*                              20
*label_number_abs*                      21
*position_likert*                       22
*signif_stars*                          24
*stat_cross*                            25
*stat_prop*                             28
*stat_weighted_mean*                    31
*weighted.median*                       33

Index

---

*augment_chisq_add_phi*  Augment a chi-squared test and compute phi coefficients

### Description

Augment a chi-squared test and compute phi coefficients

### Usage

`augment_chisq_add_phi(x)`

### Arguments

- `x`  a chi-squared test as returned by `stats::chisq.test()`

### Details

Phi coefficients are a measurement of the degree of association between two binary variables.

- A value between -1.0 to -0.7 indicates a strong negative association.
- A value between -0.7 to -0.3 indicates a weak negative association.
- A value between -0.3 to +0.3 indicates a little or no association.
- A value between +0.3 to +0.7 indicates a weak positive association.
- A value between +0.7 to +1.0 indicates a strong positive association.

### Value

A tibble.

### See Also

*stat_cross()* , `GDAtools::phi.table()` or `psych::phi()`
Examples

```r
tab <- xtabs(Freq ~ Sex + Class, data = as.data.frame(Titanic))
augment_chisq_add_phi(chisq.test(tab))
```

---

**geom_stripped_rows**  
Alternating Background Color

**Description**

Add alternating background color along the y-axis. The geom takes default aesthetics odd and even that receive color codes.

**Usage**

```r
geom_stripped_rows(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  show.legend = NA,
  inherit.aes = TRUE,
  xfrom = -Inf,
  xto = Inf,
  width = 1,
  nudge_y = 0
)
```

```r
geom_stripped_cols(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  show.legend = NA,
  inherit.aes = TRUE,
  yfrom = -Inf,
  yto = Inf,
  width = 1,
  nudge_x = 0
)
```

**Arguments**

- `mapping`  
  Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
The data to be displayed in this layer. There are three options:
If NULL, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x, 10)`).

The statistical transformation to use on the data for this layer, either as a ggproto Geom subclass or as a string naming the stat stripped of the stat_ prefix (e.g. "count" rather than "stat_count")

Position adjustment, either as a string naming the adjustment (e.g. "jitter" to use `position_jitter`), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

Logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.

If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders()`.

Limitation of the strips along the x-axis

Width of the strips

Limitation of the strips along the y-axis

Horizontal or vertical adjustment to nudge strips by

A `ggplot2` plot with the added geometry.

```
data(tips, package = "reshape")
library(ggplot2)
p <- ggplot(tips) +
aes(x = time, y = day) +
geom_count() +
theme_light()

p
p + geom_stripped_rows()
```
ggcoef_model

Plot model coefficients

Description

`ggcoef_model()`, `ggcoef_table()`, `ggcoef_multinom()`, `ggcoef_multicomponents()` and `ggcoef_compare()` use `broom.helpers::tidy_plus_plus()` to obtain a tibble of the model coefficients, apply additional data transformation and then pass the produced tibble to `ggcoef_plot()` to generate the plot.

Usage

```r
ggcoef_model(
  model,
  tidy_fun = broom.helpers::tidy_with_broom_or_parameters,
  tidy_args = NULL,
  conf.int = TRUE,
  conf.level = 0.95,
  exponentiate = FALSE,
  variable_labels = NULL,
  term_labels = NULL,
  interaction_sep = " * ",
  categorical_terms_pattern = "\{level\}",
  add_reference_rows = TRUE,
  no_reference_row = NULL,
  intercept = FALSE,
  include = dplyr::everything(),
)```
add_pairwise_contrasts = FALSE,
pairwise_variables = broom.helpers::all_categorical(),
keep_model_terms = FALSE,
pairwise_reverse = TRUE,
emmeans_args = list(),
significance = 1 - conf.level,
significance_labels = NULL,
show_p_values = TRUE,
signif_stars = TRUE,
return_data = FALSE,
...
)

ggcoef_table(  
model,
tidy_fun = broom.helpers::tidy_with_broom_or_parameters,
tidy_args = NULL,
conf.int = TRUE,
conf.level = 0.95,
exponentiate = FALSE,
variable_labels = NULL,
term_labels = NULL,
interaction_sep = " * ",
categorical_terms_pattern = "{level}",
add_reference_rows = TRUE,
no_reference_row = NULL,
intercept = FALSE,
include = dplyr::everything(),
add_pairwise_contrasts = FALSE,
pairwise_variables = broom.helpers::all_categorical(),
keep_model_terms = FALSE,
pairwise_reverse = TRUE,
emmeans_args = list(),
significance = 1 - conf.level,
significance_labels = NULL,
show_p_values = FALSE,
signif_stars = FALSE,
table_stat = c("estimate", "ci", "p.value"),
table_header = NULL,
table_text_size = 3,
table_stat_label = NULL,
ci_pattern = "[{conf.low}], [{conf.high}]",
table_widths = c(3, 2),
plot_title = NULL,
...
)

ggcoef_compare(
models,
type = c("dodged", "faceted"),
tidy_fun = broom.helpers::tidy_with_broom_or_parameters,
tidy_args = NULL,
conf.int = TRUE,
conf.level = 0.95,
exponentiate = FALSE,
variable_labels = NULL,
term_labels = NULL,
interaction_sep = " * ",
categorical_terms_pattern = "{level}",
add_reference_rows = TRUE,
no_reference_row = NULL,
intercept = FALSE,
include = dplyr::everything(),
add_pairwise_contrasts = FALSE,
pairwise_variables = broom.helpers::all_categorical(),
keep_model_terms = FALSE,
pairwise_reverse = TRUE,
emmeans_args = list(),
significance = 1 - conf.level,
significance_labels = NULL,
return_data = FALSE,
...
)


ggcoef_multinom(
  model,
  type = c("dodged", "faceted", "table"),
y.level_label = NULL,
tidy_fun = broom.helpers::tidy_with_broom_or_parameters,
tidy_args = NULL,
conf.int = TRUE,
conf.level = 0.95,
exponentiate = FALSE,
variable_labels = NULL,
term_labels = NULL,
interaction_sep = " * ",
categorical_terms_pattern = "{level}",
add_reference_rows = TRUE,
no_reference_row = NULL,
intercept = FALSE,
include = dplyr::everything(),
significance = 1 - conf.level,
significance_labels = NULL,
return_data = FALSE,
table_stat = c("estimate", "ci", "p.value"),
table_header = NULL,
ggcoef_model

```r


table_text_size = 3,
table_stat_label = NULL,
ci_pattern = "\{conf.low\}, \{conf.high\}",
table_widths = c(3, 2),

... 
}

ggcoef_multicomponents(
model,
type = c("dodged", "faceted", "table"),
component_col = "component",
component_label = NULL,
tidy_fun = broom.helpers::tidy_with_broom_or_parameters,
tidy_args = NULL,
conf.int = TRUE,
conf.level = 0.95,
exponentiate = FALSE,
variable_labels = NULL,
term_labels = NULL,
interaction_sep = " * ",
categorical_terms_pattern = "\{level\}",
add_reference_rows = TRUE,
no_reference_row = NULL,
intercept = FALSE,
include = dplyr:::everything(),
significance = 1 - conf.level,
significance_labels = NULL,
return_data = FALSE,
table_stat = c("estimate", "ci", "p.value"),
table_header = NULL,
table_text_size = 3,
table_stat_label = NULL,
ci_pattern = "\{conf.low\}, \{conf.high\}",
table_widths = c(3, 2),

... 
}

ggcoef_plot(
data,
x = "estimate",
y = "label",
exponentiate = FALSE,
point_size = 2,
point_stroke = 2,
point_fill = "white",
colour = NULL,
colour_guide = TRUE,
colour_lab = "",
```
colour_labels = ggplot2::waiver(),
shape = "significance",
shape_values = c(16, 21),
shape_guide = TRUE,
shape_lab = "",
errorbar = TRUE,
errorbar_height = 0.1,
errorbar_coloured = FALSE,
stripped_rows = TRUE,
strips_odd = "#11111111",
strips_even = "#00000000",
vline = TRUE,
vline_colour = "grey50",
dodged = FALSE,
dodged_width = 0.8,
facet_row = "var_label",
facet_col = NULL,
facet_labeller = "label_value"
)

Arguments

model a regression model object
tidy_fun option to specify a custom tidier function
tidy_args Additional arguments passed to `broom.helpers::tidy_plus_plus()` and to `tidy_fun`
conf.int should confidence intervals be computed? (see `broom::tidy()`)
conf.level the confidence level to use for the confidence interval if `conf.int = TRUE`; must be strictly greater than 0 and less than 1; defaults to 0.95, which corresponds to a 95 percent confidence interval
exponentiate if TRUE a logarithmic scale will be used for x-axis
variable_labels a named list or a named vector of custom variable labels
term_labels a named list or a named vector of custom term labels
interaction_sep separator for interaction terms
categorical_terms_pattern a glue pattern for labels of categorical terms with treatment or sum contrasts (see `model_list_terms_levels()`)
add_reference_rows should reference rows be added?
no_reference_row variables (accepts tidyselect notation) for those no reference row should be added, when `add_reference_rows = TRUE`
intercept should the intercept(s) be included?
variables to include. Accepts tidyselect syntax. Use ~ to remove a variable. Default is everything(). See also all_continuous(), all_categorical(), all_dichotomous() and all_interaction()

**add_pairwise_contrasts**
apply `tidy_add_pairwise_contrasts()`? [Experimental]

**pairwise_variables**
variables to add pairwise contrasts (accepts tidyselect notation)

**keep_model_terms**
keep original model terms for variables where pairwise contrasts are added? (default is FALSE)

**pairwise_reverse**
determines whether to use "pairwise" (if TRUE) or "revpairwise" (if FALSE), see `emmeans::contrast()`

**emmeans_args**
list of additional parameter to pass to `emmeans::emmeans()` when computing pairwise contrasts

**significance**
level (between 0 and 1) below which a coefficient is consider to be significantly different from 0 (or 1 if exponentiate = TRUE), NULL for not highlighting such coefficients

**significance_labels**
onoptional vector with custom labels for significance variable

**show_p_values**
if TRUE, add p-value to labels

**signif_stars**
if TRUE, add significant stars to labels

**return_data**
if TRUE, will return the data.frame used for plotting instead of the plot

... parameters passed to `ggcoef_plot()`

**table_stat**
statistics to display in the table, use any column name returned by the tidier or "ci" for confidence intervals formatted according to `ci_pattern`

**table_header**
onoptional custom headers for the table

**table_text_size**
text size for the table

**table_stat_label**
onoptional named list of labeller functions for the displayed statistic (see examples)

**ci_pattern**
glue pattern for confidence intervals in the table

**table_widths**
relative widths of the forest plot and the coefficients table

**plot_title**
an optional plot title

**models**
named list of models

**type**
a dodged plot, a faceted plot or multiple table plots?

**y_level_label**
an optional named vector for labeling y_level (see examples)

**component_col**
name of the component column

**component_label**
an optional named vector for labeling components

**data**
a data frame containing data to be plotted, typically the output of `ggcoef_model()`, `ggcoef_compare()` or `ggcoef_multinom()` with the option return_data = TRUE
x, y  variables mapped to x and y axis
point_size  size of the points
point_stroke  thickness of the points
point_fill  fill colour for the points
colour  optional variable name to be mapped to colour aesthetic
colour_guide  should colour guide be displayed in the legend?
colour_lab  label of the colour aesthetic in the legend
colour_labels  labels argument passed to `ggplot2::scale_colour_discrete()` and `ggplot2::discrete_scale()`
shape  optional variable name to be mapped to the shape aesthetic
shape_values  values of the different shapes to use in `ggplot2::scale_shape_manual()`
shape_guide  should shape guide be displayed in the legend?
shape_lab  label of the shape aesthetic in the legend
errorbar  should error bars be plotted?
errorbar_height  height of error bars
errorbar_coloured  should error bars be colored as the points?
stripped_rows  should stripped rows be displayed in the background?
strips_odd  color of the odd rows
strips_even  color of the even rows
vline  should a vertical line be drawn at 0 (or 1 if `exponentiate = TRUE`)?
vline_colour  colour of vertical line
dodged  should points be dodged (according to the colour aesthetic)?
dodged_width  width value for `ggplot2::position_dodge()`
facet_row  variable name to be used for row facets
facet_col  optional variable name to be used for column facets
facet_labeller  labeller function to be used for labeling facets; if labels are too long, you can use `ggplot2::label_wrap_gen()` (see examples), more information in the documentation of `ggplot2::facet_grid()`

Details

For more control, you can use the argument `return_data = TRUE` to get the produced tibble, apply any transformation of your own and then pass your customized tibble to `ggcoef_plot()`.

Value

A `ggplot2` plot or a tibble if `return_data = TRUE`. 
Functions

- `ggcoef_table()`: a variation of `ggcoef_model()` adding a table with estimates, confidence intervals and p-values.
- `ggcoef_compare()`: designed for displaying several models on the same plot.
- `ggcoef_multinom()`: a variation of `ggcoef_model()` adapted to multinomial logistic regressions performed with `nnet::multinom()`.
- `ggcoef_multicomponents()`: a variation of `ggcoef_model()` adapted to multi-component models such as zero-inflated models or beta regressions. `ggcoef_multicomponents()` has been tested with `pscl::zeroinfl()`, `pscl::hurdle()` and `betareg::betareg()`.
- `ggcoef_plot()`: plot a tidy tibble of coefficients.

See Also

vignette("ggcoef_model")

Examples

```r
mod <- lm(Sepal.Length ~ Sepal.Width + Species, data = iris)
ggcoef_model(mod)

ggcoef_table(mod)

ggcoef_table(mod, table_stat = c("estimate", "ci"))

ggcoef_table(
  mod,
  table_stat_label = list(
    estimate = scales::label_number(.001)
  )
)

ggcoef_table(mod, table_text_size = 5, table_widths = c(1, 1))

# a logistic regression example
d_titanic <- as.data.frame(Titanic)
d_titanic$Survived <- factor(d_titanic$Survived, c("No", "Yes"))
mod_titanic <- glm(
  Survived ~ Sex * Age + Class,
  weights = Freq,
  data = d_titanic,
  family = binomial
)

# use 'exponentiate = TRUE' to get the Odds Ratio
ggcoef_model(mod_titanic, exponentiate = TRUE)

ggcoef_table(mod_titanic, exponentiate = TRUE)
```
# display intercepts
ggcoef_model(mod_titanic, exponentiate = TRUE, intercept = TRUE)

# customize terms labels
ggcoef_model(
  mod_titanic,
  exponentiate = TRUE,
  show_p_values = FALSE,
  signif_stars = FALSE,
  add_reference_rows = FALSE,
  categorical_terms_pattern = "(level) (ref: {reference_level})",
  interaction_sep = " x "
) +
ggplot2::scale_y_discrete(labels = scales::label_wrap(15))

# display only a subset of terms
ggcoef_model(mod_titanic, exponentiate = TRUE, include = c("Age", "Class"))

# do not change points' shape based on significance
ggcoef_model(mod_titanic, exponentiate = TRUE, significance = NULL)

# a black and white version
ggcoef_model(
  mod_titanic,
  exponentiate = TRUE,
  colour = NULL, stripped_rows = FALSE
)

# show dichotomous terms on one row
ggcoef_model(
  mod_titanic,
  exponentiate = TRUE,
  no_reference_row = broom.helpers::all_dichotomous(),
  categorical_terms_pattern = 
    "(ifelse(dichotomous, paste0(level, '/ ', reference_level), level))",
  show_p_values = FALSE
)

data(tips, package = "reshape")
mod_simple <- lm(tip ~ day + time + total_bill, data = tips)
 ggcoef_model(mod_simple)

# custom variable labels
# you can use the labelled package to define variable labels
# before computing model
if (requireNamespace("labelled")) {
  tips_labelled <- tips %>%
    labelled::set_variable_labels(
      day = "Day of the week",
      time = "Lunch or Dinner",
    )
}
total_bill = "Bill's total"
)
mod_labelled <- lm(tip ~ day + time + total_bill, data = tips_labelled)
ggcoef_model(mod_labelled)
}

# you can provide custom variable labels with 'variable_labels'
ggcoef_model(
  mod_simple,
  variable_labels = c(
    day = "Week day",
    time = "Time (lunch or dinner?)",
    total_bill = "Total of the bill"
  ),
  facet_labeller = ggplot2::label_wrap_gen(10)
)

# if labels are too long, you can use 'facet_labeller' to wrap them
ggcoef_model(
  mod_simple,
  variable_labels = c(
    day = "Week day",
    time = "Time (lunch or dinner?)",
    total_bill = "Total of the bill"
  ),
  facet_labeller = ggplot2::label_wrap_gen(10)
)

# do not display variable facets but add colour guide
ggcoef_model(mod_simple, facet_row = NULL, colour_guide = TRUE)

# works also with with polynomial terms
mod_poly <- lm(
  tip ~ poly(total_bill, 3) + day,
  data = tips,
)
ggcoef_model(mod_poly)

# or with different type of contrasts
# for sum contrasts, the value of the reference term is computed
if (requireNamespace("emmeans")) {
  mod2 <- lm(
    tip ~ day + time + sex,
    data = tips,
    contrasts = list(time = contr.sum, day = contr.treatment(4, base = 3))
  )
ggcoef_model(mod2)
}

# Use ggcoef_compare() for comparing several models on the same plot
mod1 <- lm(Fertility ~ ., data = swiss)
mod2 <- step(mod1, trace = 0)
mod3 <- lm(Fertility ~ Agriculture + Education * Catholic, data = swiss)
models <- list(
  "Full model" = mod1,
  "Simplified model" = mod2,
  "With interaction" = mod3
)

ggcoef_compare(models)
ggcoef_compare(models, type = "faceted")

# you can reverse the vertical position of the point by using a negative
# value for dodged_width (but it will produce some warnings)
ggcoef_compare(models, dodged_width = -.9)

library(pscl)
data("bioChemists", package = "pscl")
mod <- zeroinfl(art ~ fem * mar | fem + mar, data = bioChemists)
ggcoef_multicomponents(mod)

mod2 <- zeroinfl(art ~ fem + mar | 1, data = bioChemists)
ggcoef_multicomponents(mod2, type = "t")
Description

[Experimental]

Usage

```r
gglikert(
  data,
  include = dplyr::everything(),
  weights = NULL,
  y = ".question",
  variable_labels = NULL,
  sort = c("none", "ascending", "descending"),
  sort_method = c("prop", "mean", "median"),
  sort_prop_include_center = totals_include_center,
  exclude_fill_values = NULL,
  add_labels = TRUE,
  labels_size = 3.5,
  labels_color = "black",
  labels_accuracy = 1,
  labels_hide_below = 0.05,
  add_totals = TRUE,
  totals_size = labels_size,
  totals_color = "black",
  totals_accuracy = labels_accuracy,
  totals_fontface = "bold",
  totals_include_center = FALSE,
  totals_hjust = 0.1,
  y_reverse = TRUE,
  y_label_wrap = 50,
  reverse_likert = FALSE,
  width = 0.9,
  facet_rows = NULL,
  facet_cols = NULL,
  facet_label_wrap = 50
)
```

```r
gglikert_data(
  data,
  include = dplyr::everything(),
  weights = NULL,
  variable_labels = NULL,
  sort = c("none", "ascending", "descending"),
  sort_method = c("prop", "mean", "median"),
  sort_prop_include_center = TRUE,
  exclude_fill_values = NULL
)
```

```r
gglikert_stacked(
)
```
data, include = dplyr::everything(),
weights = NULL,
y = ".question",
variable_labels = NULL,
sort = c("none", "ascending", "descending"),
sort_method = c("prop", "mean", "median"),
sort_prop_include_center = FALSE,
add_labels = TRUE,
labels_size = 3.5,
labels_color = "black",
labels_accuracy = 1,
labels_hide_below = 0.05,
add_median_line = FALSE,
y_reverse = TRUE,
y_label_wrap = 50,
reverse_fill = TRUE,
width = 0.9
)

Arguments

data a data frame
include variables to include, accept tidy-select syntax
weights optional variable name of a weighting variable, accept tidy-select syntax
y name of the variable to be plotted on y axis (relevant when .question is mapped
to "facets, see examples), accept tidy-select syntax
variable_labels a named list or a named vector of custom variable labels
sort should variables be sorted?
sort_method method used to sort the variables: "prop" sort according to the proportion of
answers higher than the centered level. "mean" considers answer as a score and
sort according to the mean score, "median" used the median and the majority
judgment rule for tie-breaking.
sort_prop_include_center when sorting with "prop" and if the number of levels is uneven, should half of
the central level be taken into account to compute the proportion?
exclude_fill_values Vector of values that should not be displayed (but still taken into account for
calculating proportions), see position_likert()
add_labels should percentage labels be added to the plot?
labels_size size of the percentage labels
labels_color color of the percentage labels
labels_accuracy accuracy of the percentages, see scales::label_percent()
labels_hide_below
if provided, values below will be masked, see \texttt{label_percent_abs()}

add_totals
should the total proportions of negative and positive answers be added to plot?
This option is not compatible with facets!

totals_size
size of the total proportions

totals_color
color of the total proportions

totals_accuracy
accuracy of the total proportions, see \texttt{scales::label_percent()}

totals_fontface
font face of the total proportions

totals_include_center
if the number of levels is uneven, should half of the center level be added to the
total proportions?

totals_hjust
horizontal adjustment of totals labels on the x axis

y_reverse
should the y axis be reversed?

y_label_wrap
number of characters per line for y axis labels, see \texttt{scales::label_wrap()}

reverse_likert
if \texttt{TRUE}, will reverse the default stacking order, see \texttt{position_likert()}

width
bar width, see \texttt{ggplot2::geom_bar()}

facet_rows, facet_cols
A set of variables or expressions quoted by \texttt{ggplot2::vars()} and defining
faceting groups on the rows or columns dimension (see examples)

facet_label_wrap
number of characters per line for facet labels, see \texttt{ggplot2::label_wrap_gen()}

add_median_line
add a vertical line at 50%?

reverse_fill
if \texttt{TRUE}, will reverse the default stacking order, see \texttt{ggplot2::position_fill()}

Details
Combines several factor variables using the same list of ordered levels (e.g. Likert-type scales) into
a unique data frame and generates a centered bar plot.

You could use \texttt{gglikert_data()} to just produce the dataset to be plotted.

If variable labels have been defined (see \texttt{labelled::var_label()}), they will be considered. You
can also pass custom variables labels with the \texttt{variable_labels} argument.

Value
A \texttt{ggplot2} plot or a \texttt{tibble}.

See Also

\texttt{vignette("gglikert"), position_likert(), stat_prop()}
Examples

library(ggplot2)
library(dplyr)

likert_levels <- c(
  "Strongly disagree",
  "Disagree",
  "Neither agree nor disagree",
  "Agree",
  "Strongly agree"
)

set.seed(42)
df <-
tibble(
  q1 = sample(likert_levels, 150, replace = TRUE),
  q2 = sample(likert_levels, 150, replace = TRUE, prob = 5:1),
  q3 = sample(likert_levels, 150, replace = TRUE, prob = 1:5),
  q4 = sample(likert_levels, 150, replace = TRUE, prob = 1:5),
  q5 = sample(c(likert_levels, NA), 150, replace = TRUE),
  q6 = sample(likert_levels, 150, replace = TRUE, prob = c(1, 0, 1, 1, 0))
) %>%
  mutate(across(everything(), ~ factor(.x, levels = likert_levels)))

gglkert(df)

gglkert(df, include = q1:3)

gglkert(df, sort = "ascending")

gglkert(df, sort = "ascending", sort_prop_include_center = TRUE)

gglkert(df, sort = "ascending", sort_method = "mean")

gglkert(df, reverse_likert = TRUE)

gglkert(df, add_totals = FALSE, add_labels = FALSE)

gglkert(
  df,
  totals_include_center = TRUE,
  totals_hjust = .25,
  totals_size = 4.5,
  totals_fontface = "italic",
  totals_accuracy = .01,
  labels_accuracy = 1,
  labels_size = 2.5,
  labels_hide_below = .25
)

gglkert(df, exclude_fill_values = "Neither agree nor disagree")
```r
if (require("labelled")) {
  df %>%
    set_variable_labels(
      q1 = "First question",
      q2 = "Second question"
    ) %>%
    gglikert(
      variable_labels = c(
        q4 = "a custom label",
        q6 = "a very very very very very very very very very very long label"
      ),
      y_label_wrap = 25
    )

  # Facets
  df_group <- df
  df_group$group <- sample(c("A", "B"), 150, replace = TRUE)
  gglikert(df_group, q1:q6, facet_rows = vars(group))
  gglikert(df_group, q1:q6, facet_cols = vars(group))
  gglikert(df_group, q1:q6, y = "group", facet_rows = vars(.question))
  gglikert_stacked(df, q1:q6)
  gglikert_stacked(df, q1:q6, add_median_line = TRUE, sort = "asc")

  gglikert_stacked(df_group, q1:q6, y = "group", add_median_line = TRUE) +
    facet_grid(rows = vars(.question))
}
```

---

**ggsurvey**

_Easy ggplot2 with survey objects_

**Description**

A function to facilitate ggplot2 graphs using a survey object. It will initiate a ggplot and map survey weights to the corresponding aesthetic.

**Usage**

`ggsurvey(design = NULL, mapping = NULL, ...)`

**Arguments**

- **design**
  
  A survey design object, usually created with `survey::svydesign()`

- **mapping**
  
  Default list of aesthetic mappings to use for plot, to be created with `ggplot2::aes()`.

- **...**
  
  Other arguments passed on to methods. Not currently used.
Details

Graphs will be correct as long as only weights are required to compute the graph. However, statistic or geometry requiring correct variance computation (like `ggplot2::geom_smooth()`) will be statistically incorrect.

Value

A `ggplot2` plot.

Examples

```r
data(api, package = "survey")
dstrat <- survey::svydesign(
  id = ~1, strata = ~stype,
  weights = ~pw, data = apistrat,
  fpc = ~fpc
)
ggsurvey(dstrat) +
  ggplot2::aes(x = cnum, y = dnum) +
  ggplot2::geom_count()

d <- as.data.frame(Titanic)
dw <- survey::svydesign(ids = ~1, weights = ~Freq, data = d)
ggsurvey(dw) +
  ggplot2::aes(x = Class, fill = Survived) +
  ggplot2::geom_bar(position = "fill")
```

---

**label_number_abs**  
*Label absolute values*

Description

Label absolute values

Usage

```r
label_number_abs(..., hide_below = NULL)
```

```r
label_percent_abs(..., hide_below = NULL)
```

Arguments

- `...` arguments passed to `scales::label_number()` or `scales::label_percent()`
- `hide_below` if provided, values below `hide_below` will be masked (i.e. an empty string "" will be returned)
See Also

`scales::label_number()`, `scales::label_percent()`

Examples

```r
x <- c(-0.2, -.05, 0, .07, .25, .66)
scales::label_number()(x)
label_number_abs()(x)
scales::label_percent()(x)
label_percent_abs()(x)
label_percent_abs(hide_below = .1)(x)
```

Description

`position_likert()` stacks proportion bars on top of each other and center them around zero (the same number of modalities are displayed on each side). This type of presentation is commonly used to display Likert-type scales. `position_likert_count()` uses counts instead of proportions.

Arguments

- `vjust` Vertical adjustment for geoms that have a position (like points or lines), not a dimension (like bars or areas). Set to 0 to align with the bottom, 0.5 for the middle, and 1 (the default) for the top.
- `reverse` If TRUE, will reverse the default stacking order. This is useful if you're rotating both the plot and legend.
- `exclude_fill_values` Vector of values from the variable associated with the fill aesthetic that should not be displayed (but still taken into account for computing proportions)

Details

It is recommended to use `position_likert()` with `stat_prop()` and its complete argument (see examples).
See Also

See `ggplot2::position_stack()` and `ggplot2::position_fill()`

Examples

```r
library(ggplot2)

ggplot(diamonds) +
aes(y = clarity, fill = cut) +
geom_bar(position = "fill") +
  scale_x_continuous(label = scales::label_percent()) +
  scale_fill_brewer(palette = "PiYG") +
  xlab("proportion")

ggplot(diamonds) +
aes(y = clarity, fill = cut) +
geom_bar(position = "likert") +
  scale_x_continuous(label = label_percent_abs()) +
  scale_fill_brewer(palette = "PiYG") +
  xlab("proportion")

ggplot(diamonds) +
aes(y = clarity, fill = cut) +
geom_bar(position = "stack") +
  scale_fill_brewer(palette = "PiYG")

ggplot(diamonds) +
aes(y = clarity, fill = cut) +
geom_bar(position = "likert_count") +
  scale_x_continuous(label = label_number_abs()) +
  scale_fill_brewer(palette = "PiYG")

# Reverse order -------------------------------------------------------------

ggplot(diamonds) +
aes(y = clarity, fill = cut) +
geom_bar(position = position_likert(reverse = TRUE)) +
  scale_x_continuous(label = label_percent_abs()) +
  scale_fill_brewer(palette = "PiYG", direction = -1) +
  xlab("proportion")

# Missing items -------------------------------------------------------------

# example with a level not being observed for a specific value of y

# by default, the two lowest bar are not properly centered

ggplot(d) +
aes(y = clarity, fill = cut) +
geom_bar(position = "likert") +
```
scale_fill_brewer(palette = "PiYG")

# use stat_prop() with `complete = "fill"` to fix it
ggplot(d) +
aes(y = clarity, fill = cut) +
  geom_bar(position = "likert", stat = "prop", complete = "fill") +
  scale_fill_brewer(palette = "PiYG")

# Add labels -----------------------------------------------
custom_label <- function(x) {
  p <- scales::percent(x, accuracy = 1)
  p[x < .075] <- ""
  p
}

ggplot(diamonds) +
aes(y = clarity, fill = cut) +
  geom_bar(position = "likert") +
  geom_text(
    aes(by = clarity, label = custom_label(after_stat(prop))),
    stat = "prop",
    position = position_likert(vjust = .5)
  ) +
  scale_x_continuous(label = label_percent_abs()) +
  scale_fill_brewer(palette = "PiYG", direction = -1) +
  xlab("proportion")

# Do not display specific fill values -----------------------------------
# (but taken into account to compute proportions)

ggplot(diamonds) +
aes(y = clarity, fill = cut) +
  geom_bar(position = position_likert(exclude_fill_values = "Very Good")) +
  scale_x_continuous(label = label_percent_abs()) +
  scale_fill_brewer(palette = "PiYG") +
  xlab("proportion")

---

signif_stars | Significance Stars

Description

Calculate significance stars

Usage

signif_stars(x, three = 0.001, two = 0.01, one = 0.05, point = 0.1)
Arguments

- **x**: numeric values that will be compared to the point, one, two, and three values
- **three**: threshold below which to display three stars
- **two**: threshold below which to display two stars
- **one**: threshold below which to display one star
- **point**: threshold below which to display one point (NULL to deactivate)

Value

Character vector containing the appropriate number of stars for each x value.

Author(s)

Joseph Larmarange

Examples

```r
x <- c(0.5, 0.1, 0.05, 0.01, 0.001)
signif_stars(x)
signif_stars(x, one = .15, point = NULL)
```

Description

Computes statistics of a 2-dimensional matrix using `broom::augment.htest`.

Usage

```r
stat_cross(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  keep.zero.cells = FALSE
)
```
Arguments

mapping Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.

data The data to be displayed in this layer. There are three options:
If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot().
A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.
A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).

geom Override the default connection with ggplot2::geom_point().
position Position adjustment, either as a string naming the adjustment (e.g. "jitter" to use position_jitter), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

... Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.

na.rm If TRUE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
show.legend logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. borders().
keep.zero.cells If TRUE, cells with no observations are kept.

Value
A ggplot2 plot with the added statistic.

Aesthetics
stat_cross() requires the x and the y aesthetics.

Computed variables

observed number of observations in x,y
prop proportion of total
row.prop row proportion
col.prop column proportion
**stat_cross**

expected  expected count under the null hypothesis  
resid  Pearson’s residual  
std.resid  standardized residual  
row.observed  total number of observations within row  
col.observed  total number of observations within column  
total.observed  total number of observations within the table  
phi  phi coefficients, see augment_chisq_add_phi()

See Also

vignette("stat_cross")

Examples

```r
library(ggplot2)
d <- as.data.frame(Titanic)

# plot number of observations
ggplot(d) +
aes(x = Class, y = Survived, weight = Freq, size = after_stat(observed)) +
stat_cross() +
scale_size_area(max_size = 20)

# custom shape and fill colour based on chi-squared residuals
ggplot(d) +
aes(
  x = Class, y = Survived, weight = Freq,
  size = after_stat(observed), fill = after_stat(std.resid)
) +
stat_cross(shape = 22) +
scale_fill_steps2(breaks = c(-3, -2, 2, 3), show.limits = TRUE) +
scale_size_area(max_size = 20)

# custom shape and fill colour based on phi coefficients
ggplot(d) +
aes(
  x = Class, y = Survived, weight = Freq,
  size = after_stat(observed), fill = after_stat(phi)
) +
stat_cross(shape = 22) +
scale_fill_steps2(show.limits = TRUE) +
scale_size_area(max_size = 20)

# plotting the number of observations as a table
ggplot(d) +
aes(
  x = Class, y = Survived, weight = Freq, label = after_stat(observed)
) +
```
stat_prop

Compute proportions according to custom denominator

Description

stat_prop() is a variation of ggplot2::stat_count() allowing to compute custom proportions according to the by aesthetic defining the denominator (i.e. all proportions for a same value of by will sum to 1). The by aesthetic should be a factor. If by is not specified, proportions of the total will be computed.

Usage

stat_prop(
  mapping = NULL,
  data = NULL,
  geom = "bar",
  position = "fill",
  ..., 
  width = NULL,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE,
  complete = NULL
)

Arguments

mapping Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
stat_prop

data The data to be displayed in this layer. There are three options:
If NULL, the default, the data is inherited from the plot data as specified in the
call to ggplot().
A data.frame, or other object, will override the plot data. All objects will be
fortified to produce a data frame. See fortify() for which variables will be
created.
A function will be called with a single argument, the plot data. The return
value must be a data.frame, and will be used as the layer data. A function
can be created from a formula (e.g. ~ head(.x, 10)).

geom Override the default connection with ggplot2::geom_bar().

position Position adjustment, either as a string naming the adjustment (e.g. "jitter" to
use position_jitter), or the result of a call to a position adjustment function.
Use the latter if you need to change the settings of the adjustment.

... Other arguments passed on to layer(). These are often aesthetics, used to set
an aesthetic to a fixed value, like colour = "red" or size = 3. They may also
be parameters to the paired geom/stat.

width Bar width. By default, set to 90% of the resolution() of the data.

na.rm If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

orientation The orientation of the layer. The default (NA) automatically determines the ori-
entation from the aesthetic mapping. In the rare event that this fails it can be
given explicitly by setting orientation to either "x" or "y". See the Orientation
section for more detail.

show.legend logical. Should this layer be included in the legends? NA, the default, includes if
any aesthetics are mapped. FALSE never includes, and TRUE always includes. It
can also be a named logical vector to finely select the aesthetics to display.

inherit.aes If FALSE, overrides the default aesthetics, rather than combining with them.
This is most useful for helper functions that define both data and aesthetics and
shouldn’t inherit behaviour from the default plot specification, e.g. borders().

complete Name (character) of an aesthetic for those statistics should be completed for
unobserved values (see example)

Value
A ggplot2 plot with the added statistic.

Aesthetics
stat_prop() understands the following aesthetics (required aesthetics are in bold):

• x or y
• by (this aesthetic should be a factor)
• group
• weight
Computed variables

- **count**: number of points in bin
- **prop**: computed proportion

See Also

- vignette("stat_prop"), `ggplot2::stat_count()`

Examples

```r
library(ggplot2)
d <- as.data.frame(Titanic)
p <- ggplot(d) +
  aes(x = Class, fill = Survived, weight = Freq, by = Class) +
  geom_bar(position = "fill") +
  geom_text(stat = "prop", position = position_fill(.5))
p
p + facet_grid(~Sex)
```

```r
ggplot(d) +
  aes(x = Class, fill = Survived, weight = Freq) +
  geom_bar(position = "dodge") +
  geom_text(
    aes(by = Survived),
    stat = "prop",
    position = position_dodge(0.9), vjust = "bottom"
  )
```

```r
if (requireNamespace("scales")) {
  ggplot(d) +
  aes(x = Class, fill = Survived, weight = Freq, by = 1) +
  geom_bar() +
  geom_text(
    aes(label = scales::percent(after_stat(prop), accuracy = 1)),
    stat = "prop",
    position = position_stack(.5)
  )
}
```

# displaying unobserved levels with complete
```r
d <- diamonds %>%
  dplyr::select(-c(cut, clarity)) %>%
  dplyr::filter(!(cut == "Ideal" & clarity == "I1")) %>%
  dplyr::filter(!(cut == "Very Good" & clarity == "VS2")) %>%
  dplyr::filter(!(cut == "Premium" & clarity == "IF"))
p <- ggplot(d) +
  aes(x = clarity, fill = cut, by = clarity) +
  geom_bar(position = "fill")
p + geom_text(stat = "prop", position = position_fill(.5))
p + geom_text(stat = "prop", position = position_fill(.5), complete = "fill")
```
stat_weighted_mean

Compute weighted y mean

Description

This statistic will compute the mean of y aesthetic for each unique value of x, taking into account weight aesthetic if provided.

Usage

stat_weighted_mean(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)

Arguments

mapping Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.

data The data to be displayed in this layer. There are three options:
If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot().
A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.
A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).

geom Override the default connection with ggplot2::geom_point().

position Position adjustment, either as a string naming the adjustment (e.g. "jitter" to use position_jitter), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

... Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.

na.rm If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
orientation  The orientation of the layer. The default (NA) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be given explicitly by setting orientation to either "x" or "y". See the Orientation section for more detail.

show.legend logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. borders().

Value
A ggplot2 plot with the added statistic.

Computed variables
- y weighted y (numerator / denominator)
- numerator numerator
- denominator denominator

See Also
vignette("stat_weighted_mean")

Examples

library(ggplot2)

data(tips, package = "reshape")

ggplot(tips) +
aes(x = day, y = total_bill) +
geom_point()

ggplot(tips) +
aes(x = day, y = total_bill) +
stat_weighted_mean()

ggplot(tips) +
aes(x = day, y = total_bill, group = 1) +
stat_weighted_mean(geom = "line")

ggplot(tips) +
aes(x = day, y = total_bill, colour = sex, group = sex) +
stat_weighted_mean(geom = "line")
GGPLOT(TIPS) +
  AES(X = DAY, Y = TOTAL_BILL, FILL = SEX) +
  STAT_WEIGHTED_MEAN(GEOM = "BAR", POSITION = "DODGE")

# computing a proportion on the fly
IF (REQUIRENAMESPACE("SCALES")) {
  GGPLOT(TIPS) +
    AES(X = DAY, Y = AS.INTEGER(SMOKER == "YES"), FILL = SEX) +
    STAT_WEIGHTED_MEAN(GEOM = "BAR", POSITION = "DODGE") +
    SCALE_Y_CONTINUOUS(LABELS = SCALES::PERCENT)
}

LIBRARY(GGPPLOT2)

# taking into account some weights
IF (REQUIRENAMESPACE("SCALES")) {
  D <- AS.DATASETFRAME(TITANIC)
  GGPLOT(D) +
    AES(
      X = CLASS, Y = AS.INTEGER(SURVIVED == "YES"),
      WEIGHT = FREQ, FILL = SEX
    ) +
    GEOM_BAR(STAT = "WEIGHTED_MEAN", POSITION = "DODGE") +
    SCALE_Y_CONTINUOUS(LABELS = SCALES::PERCENT) +
    LABS(Y = "SURVIVED")
}

---

**Weighted Median and Quantiles**

**Description**

Compute the median or quantiles a set of numbers which have weights associated with them.

**Usage**

```r
weighted.median(x, w, na.rm = TRUE, type = 2)
weighted.quantile(x, w, probs = seq(0, 1, 0.25), na.rm = TRUE, type = 4)
```

**Arguments**

- `x` : a numeric vector of values
- `w` : a numeric vector of weights
- `na.rm` : a logical indicating whether to ignore NA values
- `type` : Integer specifying the rule for calculating the median or quantile, corresponding to the rules available for `stats:quantile()`. The only valid choices are type=1, 2 or 4. See Details.
- `probs` : probabilities for which the quantiles should be computed, a numeric vector of values between 0 and 1
Details

The \( i \)th observation \( x[i] \) is treated as having a weight proportional to \( w[i] \).

The weighted median is a value \( m \) such that the total weight of data less than or equal to \( m \) is equal to half the total weight. More generally, the weighted quantile with probability \( p \) is a value \( q \) such that the total weight of data less than or equal to \( q \) is equal to \( p \) times the total weight.

If there is no such value, then

- if \( \text{type} = 1 \), the next largest value is returned (this is the right-continuous inverse of the left-continuous cumulative distribution function);
- if \( \text{type} = 2 \), the average of the two surrounding values is returned (the average of the right-continuous and left-continuous inverses);
- if \( \text{type} = 4 \), linear interpolation is performed.

Note that the default rule for \( \text{weighted.median()} \) is \( \text{type} = 2 \), consistent with the traditional definition of the median, while the default for \( \text{weighted.quantile()} \) is \( \text{type} = 4 \).

Source

These functions are adapted from their homonyms developed by Adrian Baddeley in the \texttt{spatstat} package.

Examples

```r
x <- 1:20
w <- runif(20)
weighted.median(x, w)
weighted.quantile(x, w)
```
Index

* datasets
  position_likert, 22
  stat_cross, 25
  stat_prop, 28
  stat_weighted_mean, 31
  aes(), 3, 26, 28, 31
  all_categorical(), 10
  all_continuous(), 10
  all_dichotomous(), 10
  all_interaction(), 10
  augment_chisq_add_phi, 2
  augment_chisq_add_phi(), 27
  borders(), 4, 26, 29, 32
  broom::tidy_plus_plus(), 5, 9
  broom::tidy(), 9
  emmeans::contrast(), 10
  emmeans::emmeans(), 10
  fortify(), 4, 26, 29, 31
  geom_stripped_cols
    (geom_stripped_rows), 3
  geom_stripped_rows, 3
  ggcoef_compare (ggcoef_model), 5
  ggcoef_model, 5
  ggcoef_model(), 12
  ggcoef_multicomponents (ggcoef_model), 5
  ggcoef_multicomponents(), 12
  ggcoef_multinom (ggcoef_model), 5
  ggcoef_plot (ggcoef_model), 5
  ggcoef_plot(), 10
  ggcoef_table (ggcoef_model), 5
  gllikert, 15
  gglikert_data(gllikert), 15
  gglikert_stacked (gllikert), 15
  ggplot(), 4, 26, 29, 31
  ggplot2::aes(), 20
  ggplot2::discrete_scale(), 11
  ggplot2::facet_grid(), 11
  ggplot2::geom_bar(), 18, 29
  ggplot2::geom_point(), 26, 31
  ggplot2::geom_smooth(), 21
  ggplot2::label_wrap_gen(), 11, 18
  ggplot2::position_dodge(), 11
  ggplot2::position_fill(), 18, 23
  ggplot2::position_stack(), 23
  ggplot2::scale_colour_discrete(), 11
  ggplot2::scale_shape_manual(), 11
  ggplot2::stat_count(), 28, 30
  ggplot2::vars(), 18
  ggsurvey, 20
  glue pattern, 9
  label_number.abs, 21
  label_percent.abs (label_number.abs), 21
  label_percent.abs(), 18
  labelled::var_label(), 18
  layer(), 4, 26, 29, 31
  model_list_terms_levels(), 9
  nnet::multinom(), 12
  position_likert, 22
  position_likert(), 17, 18
  position_likert_count
    (position_likert), 22
  PositionLikert (position_likert), 22
  PositionLikertCount (position_likert), 22
  resolution(), 29
  scales::label_number(), 21, 22
  scales::label_percent(), 17, 18, 21, 22
  scales::label_wrap(), 18
  signif.stars, 17, 18, 22
  stat_cross, 25
stat_cross(), 2
stat_prop, 28
stat_prop(), 18
stat_weighted_mean, 31
StatCross(stat_cross), 25
StatProp(stat_prop), 28
stats::chisq.test(), 2
StatWeightedMean(stat_weighted_mean), 31
survey::svydesign(), 20

tidy-select, 17
tidy_add_pairwise_contrasts(), 10
tidyselect, 9, 10

weighted.median, 33
weighted.quantile(weighted.median), 33