

Package ‘ggstats’

September 2, 2026

Type Package

Title Extension to 'ggplot2' for Plotting Stats

Version 0.14.0

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)

URL <https://larmarange.github.io/ggstats/>,
<https://github.com/larmarange/ggstats>

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

Depends R (>= 4.2)

Imports cli, dplyr, forcats, ggplot2 (>= 4.0.0), lifecycle, patchwork, purrr, rlang, scales, stats, stringr, utils, tidyr

Suggests betareg, broom, broom.helpers (>= 1.20.0), emmeans, glue, gtsummary, knitr, labelled (>= 2.11.0), reshape, rmarkdown, nnet, parameters, pscl, testthat (>= 3.0.0), spelling, survey, survival, vdiff

Encoding UTF-8

Config/testthat/edition 3

Language en-US

VignetteBuilder knitr

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 2026-09-02 05:20:33 UTC

Contents

augment_chisq_add_phi	2
geom_connector	3
geom_diverging	7
geom_prop_bar	9
geom_stripped_rows	12
ggcascade	15
ggcoef_model	17
ggcoef_multicomponents	27
gglikert	30
ggsurvey	37
hex_bw	38
label_number_abs	39
pal_extender	40
position_likert	41
round_any	44
scale_fill_likert	45
signif_stars	46
stat_cross	47
stat_prop	50
stat_weighted_mean	54
symmetric_limits	57
weighted.median	58
weighted.sum	59
Index	60

augment_chisq_add_phi	<i>Augment a chi-squared test and compute phi coefficients</i>
-----------------------	--

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

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

geom_connector	<i>Connect bars / points</i>
----------------	------------------------------

Description

`geom_connector()` is a variation of [ggplot2::geom_step\(\)](#). Its variant `geom_bar_connector()` is particularly adapted to connect bars.

Usage

```
geom_connector(  
  mapping = NULL,  
  data = NULL,  
  stat = "identity",  
  position = "identity",  
  width = 0.1,  
  continuous = FALSE,  
  na.rm = FALSE,  
  orientation = NA,  
  show.legend = NA,  
  inherit.aes = TRUE,  
  ...  
)
```

```
geom_bar_connector(
  mapping = NULL,
  data = NULL,
  stat = "prop",
  position = "stack",
  width = 0.9,
  continuous = FALSE,
  add_baseline = TRUE,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes() . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to ggplot(). A <code>data.frame</code>, 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 <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).
stat	The statistical transformation to use on the data for this layer. When using a <code>geom_*()</code> function to construct a layer, the <code>stat</code> argument can be used to override the default coupling between geoms and stats. The <code>stat</code> argument accepts the following: • A <code>Stat</code> ggproto subclass, for example <code>StatCount</code>. • A string naming the stat. To give the stat as a string, strip the function name of the <code>stat_</code> prefix. For example, to use <code>stat_count()</code>, give the stat as <code>"count"</code>. • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The <code>position</code> argument accepts the following: • The result of calling a position function, such as <code>position_jitter()</code>. This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <code>position_jitter()</code>, give the position as <code>"jitter"</code>.

	For more information and other ways to specify the position, see the layer position documentation.
width	Bar width (see examples).
continuous	Should connect segments be continuous?
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 <i>Orientation</i> 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. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
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. annotation_borders() .
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can <i>not</i> be passed through ... Unknown arguments that are not part of the 4 categories below are ignored. - Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an Aesthetics section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data. - When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept. - Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept. - The key_glyph argument of layer() may also be passed on through This can be one of the functions described as key glyphs, to change the display of the layer in the legend.
add_baseline	Add connectors at baseline?

Examples

```
library(ggplot2)
```

```

# geom_bar_connector() -----

ggplot(diamonds) +
  aes(x = clarity, fill = cut) +
  geom_bar(width = .5) +
  geom_bar_connector(width = .5, linewidth = .25) +
  theme_minimal() +
  theme(legend.position = "bottom")

ggplot(diamonds) +
  aes(x = clarity, fill = cut) +
  geom_bar(width = .5) +
  geom_bar_connector(
    width = .5,
    continuous = TRUE,
    colour = "red",
    linetype = "dotted",
    add_baseline = FALSE,
  ) +
  theme(legend.position = "bottom")

ggplot(diamonds) +
  aes(x = clarity, fill = cut) +
  geom_bar(width = .5, position = "fill") +
  geom_bar_connector(width = .5, position = "fill") +
  theme(legend.position = "bottom")

ggplot(diamonds) +
  aes(x = clarity, fill = cut) +
  geom_bar(width = .5, position = "diverging") +
  geom_bar_connector(width = .5, position = "diverging", linewidth = .25) +
  theme(legend.position = "bottom")

# geom_connector() -----

ggplot(mtcars) +
  aes(x = wt, y = mpg, colour = factor(cyl)) +
  geom_connector() +
  geom_point()

ggplot(mtcars) +
  aes(x = wt, y = mpg, colour = factor(cyl)) +
  geom_connector(continuous = TRUE) +
  geom_point()

ggplot(mtcars) +
  aes(x = wt, y = mpg, colour = factor(cyl)) +
  geom_connector(continuous = TRUE, width = .3) +
  geom_point()

ggplot(mtcars) +
  aes(x = wt, y = mpg, colour = factor(cyl)) +

```

```
geom_connector(width = 0) +  
geom_point()  
  
ggplot(mtcars) +  
  aes(x = wt, y = mpg, colour = factor(cyl)) +  
  geom_connector(width = Inf) +  
  geom_point()  
  
ggplot(mtcars) +  
  aes(x = wt, y = mpg, colour = factor(cyl)) +  
  geom_connector(width = Inf, continuous = TRUE) +  
  geom_point()
```

geom_diverging

Geometries for diverging bar plots

Description

These geometries are variations of `ggplot2::geom_bar()` and `ggplot2::geom_text()` but provides different set of default values.

Usage

```
geom_diverging(  
  mapping = NULL,  
  data = NULL,  
  position = "diverging",  
  ...,  
  complete = "fill",  
  default_by = "total"  
)
```

```
geom_likert(  
  mapping = NULL,  
  data = NULL,  
  position = "likert",  
  ...,  
  complete = "fill",  
  default_by = "x"  
)
```

```
geom_pyramid(  
  mapping = NULL,  
  data = NULL,  
  position = "diverging",  
  ...,  
  complete = NULL,
```

```

    default_by = "total"
  )

geom_diverging_text(
  mapping = ggplot2::aes(!!!auto_contrast),
  data = NULL,
  position = position_diverging(0.5),
  ...,
  complete = "fill",
  default_by = "total"
)

geom_likert_text(
  mapping = ggplot2::aes(!!!auto_contrast),
  data = NULL,
  position = position_likert(0.5),
  ...,
  complete = "fill",
  default_by = "x"
)

geom_pyramid_text(
  mapping = ggplot2::aes(!!!auto_contrast),
  data = NULL,
  position = position_diverging(0.5),
  ...,
  complete = NULL,
  default_by = "total"
)

```

Arguments

mapping	Optional set of aesthetic mappings.
data	The data to be displayed in this layers.
position	A position adjustment to use on the data for this layer.
...	Other arguments passed on to ggplot2::geom_bar()
complete	An aesthetic for those unobserved values should be completed, see stat_prop() .
default_by	Name of an aesthetic determining denominators by default, see stat_prop() .

Details

- `geom_diverging()` is designed for stacked diverging bar plots, using [position_diverging\(\)](#).
- `geom_likert()` is designed for Likert-type items. Using [position_likert\(\)](#) (each bar sums to 100%).
- `geom_pyramid()` is similar to `geom_diverging()` but uses proportions of the total instead of counts.

To add labels on the bar plots, simply use `geom_diverging_text()`, `geom_likert_text()`, or `geom_pyramid_text()`.

All these geometries relies on `stat_prop()`.

Examples

```
library(ggplot2)
ggplot(diamonds) +
  aes(x = clarity, fill = cut) +
  geom_diverging()

ggplot(diamonds) +
  aes(x = clarity, fill = cut) +
  geom_diverging(position = position_diverging(cutoff = 4))

ggplot(diamonds) +
  aes(y = clarity, fill = cut) +
  geom_likert() +
  geom_likert_text()

ggplot(diamonds) +
  aes(y = clarity, fill = cut) +
  geom_likert() +
  geom_likert_text(
    aes(
      label = label_percent_abs(accuracy = 1, hide_below = .10)(
        after_stat(prop)
      ),
      colour = after_scale(hex_bw(.data$fill))
    )
  )

d <- Titanic |> as.data.frame()

ggplot(d) +
  aes(y = Class, fill = Sex, weight = Freq) +
  geom_diverging() +
  geom_diverging_text()

ggplot(d) +
  aes(y = Class, fill = Sex, weight = Freq) +
  geom_pyramid() +
  geom_pyramid_text()
```

Description

`geom_prop_bar()`, `geom_prop_text()` and `geom_prop_connector()` are variations of `ggplot2::geom_bar()`, `ggplot2::geom_text()` and `geom_bar_connector()` using `stat_prop()`, with custom default aesthetics: `after_stat(prop)` for **x** or **y**, and `scales::percent(after_stat(prop))` for **label**.

Usage

```
geom_prop_bar(
  mapping = NULL,
  data = NULL,
  position = "stack",
  ...,
  width = 0.9,
  complete = NULL,
  default_by = "x"
)

geom_prop_text(
  mapping = ggplot2::aes(!!!auto_contrast),
  data = NULL,
  position = ggplot2::position_stack(0.5),
  ...,
  complete = NULL,
  default_by = "x"
)

geom_prop_connector(
  mapping = NULL,
  data = NULL,
  position = "stack",
  ...,
  width = 0.9,
  complete = "fill",
  default_by = "x"
)
```

Arguments

mapping	Set of aesthetic mappings created by <code>aes()</code> . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code>. A <code>data.frame</code>, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created.

	A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as <code>position_jitter()</code>. This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <code>position_jitter()</code>, give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Additional parameters passed to <code>ggplot2::geom_bar()</code> , <code>ggplot2::geom_text()</code> or <code>geom_bar_connector()</code> .
width	Bar width (0.9 by default).
complete	Name (character) of an aesthetic for those statistics should be completed for unobserved values (see example).
default_by	If the by aesthetic is not available, name of another aesthetic that will be used to determine the denominators (e.g. "fill"), or NULL or "total" to compute proportions of the total. To be noted, <code>default_by = "x"</code> works both for vertical and horizontal bars.

See Also

[geom_bar_connector\(\)](#)

Examples

```
library(ggplot2)
d <- as.data.frame(Titanic)
ggplot(d) +
  aes(x = Class, fill = Survived, weight = Freq) +
  geom_prop_bar() +
  geom_prop_text() +
  geom_prop_connector()

ggplot(d) +
  aes(y = Class, fill = Survived, weight = Freq) +
  geom_prop_bar(width = .5) +
  geom_prop_text() +
  geom_prop_connector(width = .5, linetype = "dotted")

ggplot(d) +
  aes(
    x = Class,
    fill = Survived,
    weight = Freq,
```

```
    y = after_stat(count),  
    label = after_stat(count)  
  ) +  
  geom_prop_bar() +  
  geom_prop_text() +  
  geom_prop_connector()
```

geom_stripped_rows	<i>Alternating Background Color</i>
--------------------	-------------------------------------

Description

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

Usage

```
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  
)
```

```
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 <code>inherit.aes = TRUE</code> (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 <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to ggplot(). A <code>data.frame</code>, 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 <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).
stat	The statistical transformation to use on the data for this layer. When using a <code>geom_*()</code> function to construct a layer, the <code>stat</code> argument can be used to override the default coupling between geoms and stats. The <code>stat</code> argument accepts the following: • A Stat ggproto subclass, for example <code>StatCount</code>. • A string naming the stat. To give the stat as a string, strip the function name of the <code>stat_</code> prefix. For example, to use <code>stat_count()</code>, give the stat as <code>"count"</code>. • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The <code>position</code> argument accepts the following: • The result of calling a position function, such as <code>position_jitter()</code>. This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <code>position_jitter()</code>, give the position as <code>"jitter"</code>. • For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to layer()'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the <code>position</code> argument, or aesthetics that are required can <i>not</i> be passed through ... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an Aesthetics section that lists the available options. The 'required' aesthetics cannot be passed on to the <code>params</code>. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.

- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The `geom`'s documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The `stat`'s documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as [key glyphs](#), to change the display of the layer in the legend.

<code>show.legend</code>	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. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
<code>inherit.aes</code>	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. annotation_borders() .
<code>xfrom, xto</code>	limitation of the strips along the x-axis
<code>width</code>	width of the strips
<code>yfrom, yto</code>	limitation of the strips along the y-axis
<code>nudge_x, nudge_y</code>	horizontal or vertical adjustment to nudge strips by

Value

A `ggplot2` plot with the added geometry.

Examples

```
data(tips, package = "reshape")

library(ggplot2)
p <- ggplot(tips) +
  aes(x = time, y = day) +
  geom_count() +
  theme_light()

p
p + geom_stripped_rows()
p + geom_stripped_cols()
p + geom_stripped_rows() + geom_stripped_cols()

p <- ggplot(tips) +
  aes(x = total_bill, y = day) +
  geom_count() +
```

```

    theme_light()
  p
  p + geom_stripped_rows()
  p + geom_stripped_rows() + scale_y_discrete(expand = expansion(0, 0.5))
  p + geom_stripped_rows(xfrom = 10, xto = 35)
  p + geom_stripped_rows(odd = "blue", even = "yellow")
  p + geom_stripped_rows(odd = "blue", even = "yellow", alpha = .1)
  p + geom_stripped_rows(odd = "#00FF0022", even = "#FF000022")

  p + geom_stripped_cols()
  p + geom_stripped_cols(width = 10)
  p + geom_stripped_cols(width = 10, nudge_x = 5)

```

ggccascade

Cascade plot

Description

[Experimental]

Usage

```

ggccascade(
  .data,
  ...,
  .weights = NULL,
  .by = NULL,
  .nrow = NULL,
  .ncol = NULL,
  .add_n = TRUE,
  .text_size = 4,
  .arrows = TRUE
)

compute_cascade(.data, ..., .weights = NULL, .by = NULL)

plot_cascade(
  .data,
  .by = NULL,
  .nrow = NULL,
  .ncol = NULL,
  .add_n = TRUE,
  .text_size = 4,
  .arrows = TRUE
)

```

Arguments

<code>.data</code>	A data frame, or data frame extension (e.g. a tibble). For <code>plot_cascade()</code> , the variable displayed on the x-axis should be named "x" and the number of observations should be named "n", like the tibble returned by <code>compute_cascade()</code> .
<code>...</code>	<data-masking> Name-value pairs of conditions defining the different statuses to be plotted (see examples).
<code>.weights</code>	<tidy-select> Optional weights. Should select only one variable.
<code>.by</code>	<tidy-select> A variable or a set of variables to group by the computation of the cascade, and to generate facets. To select several variables, use <code>dplyr::pick()</code> (see examples).
<code>.nrow, .ncol</code>	Number of rows and columns, for faceted plots.
<code>.add_n</code>	Display the number of observations?
<code>.text_size</code>	Size of the labels, passed to <code>ggplot2::geom_text()</code> .
<code>.arrows</code>	Display arrows between statuses?

Details

`ggcascade()` calls `compute_cascade()` to generate a data set passed to `plot_cascade()`. Use `compute_cascade()` and `plot_cascade()` for more controls.

Value

A `ggplot2` plot or a tibble.

Examples

```
ggplot2::diamonds |>
  ggcascade(
    all = TRUE,
    big = carat > .5,
    "big & ideal" = carat > .5 & cut == "Ideal"
  )

ggplot2::mpg |>
  ggcascade(
    all = TRUE,
    recent = year > 2000,
    "recent & economic" = year > 2000 & displ < 3,
    .by = cyl,
    .ncol = 3,
    .arrows = FALSE,
    .text_size = 3
  )

ggplot2::mpg |>
  ggcascade(
    all = TRUE,
    recent = year > 2000,
```

```

    "recent & economic" = year > 2000 & displ < 3,
    .by = pick(cyl, drv),
    .add_n = FALSE,
    .text_size = 2
  )

```

ggcoef_model

Plot model coefficients

Description

ggcoef_model(), ggcoef_table(), ggcoef_dodged(), ggcoef_faceted() 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

```

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(),
  group_by = broom.helpers::auto_group_by(),
  group_labels = NULL,
  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(),
  group_by = broom.helpers::auto_group_by(),
  group_labels = NULL,
  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),
  table_witdhs = deprecated(),
  ...
)

```

```

ggcoef_dodged(
  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(),
    group_by = broom.helpers::auto_group_by(),
    group_labels = NULL,
    significance = 1 - conf.level,
    significance_labels = NULL,
    return_data = FALSE,
    ...
  )

ggcoef_faceted(
  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(),
  group_by = broom.helpers::auto_group_by(),
  group_labels = NULL,
  significance = 1 - conf.level,
  significance_labels = NULL,
  return_data = FALSE,
  ...
)

ggcoef_compare(
  models,
  type = c("dodged", "faceted", "table"),
  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,
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),
return_data = FALSE,
...
)

```

```

ggcoef_plot(
  data,
  x = "estimate",
  y = "label",
  exponentiate = FALSE,
  y_labeller = NULL,
  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",
    plot_title = NULL,
    x_limits = NULL
  )

```

Arguments

<code>model</code>	a regression model object
<code>tidy_fun</code>	(function) Option to specify a custom tidier function.
<code>tidy_args</code>	Additional arguments passed to <code>broom.helpers::tidy_plus_plus()</code> and to <code>tidy_fun</code>
<code>conf.int</code>	(logical) Should confidence intervals be computed? (see <code>broom::tidy()</code>)
<code>conf.level</code>	the confidence level to use for the confidence interval if <code>conf.int = TRUE</code> ; must be strictly greater than 0 and less than 1; defaults to 0.95, which corresponds to a 95 percent confidence interval
<code>exponentiate</code>	if TRUE a logarithmic scale will be used for x-axis
<code>variable_labels</code>	(formula-list-selector) A named list or a named vector of custom variable labels.
<code>term_labels</code>	(list or vector) A named list or a named vector of custom term labels.
<code>interaction_sep</code>	(string) Separator for interaction terms.
<code>categorical_terms_pattern</code>	(blue pattern) A blue pattern for labels of categorical terms with treatment or sum contrasts (see <code>model_list_terms_levels()</code>).
<code>add_reference_rows</code>	(logical) Should reference rows be added?
<code>no_reference_row</code>	(tidy-select) Variables for those no reference row should be added, when <code>add_reference_rows = TRUE</code> .
<code>intercept</code>	(logical) Should the intercept(s) be included?
<code>include</code>	(tidy-select) Variables to include. Default is <code>everything()</code> . See also <code>all_continuous()</code> , <code>all_categorical()</code> , <code>all_dichotomous()</code> and <code>all_interaction()</code> .
<code>group_by</code>	(tidy-select) One or several variables to group by. Default is <code>auto_group_by()</code> . Use NULL to force ungrouping.

group_labels	(string) An optional named vector of custom term labels.
add_pairwise_contrasts	(logical) Apply <code>tidy_add_pairwise_contrasts()</code> ?
pairwise_variables	(<code>tidy-select</code>) Variables to add pairwise contrasts.
keep_model_terms	(logical) Keep original model terms for variables where pairwise contrasts are added? (default is FALSE)
pairwise_reverse	(logical) Determines whether to use "pairwise" (if TRUE) or "revpairwise" (if FALSE), see <code>emmeans::contrast()</code> .
emmeans_args	(list) List of additional parameter to pass to <code>emmeans::emmeans()</code> 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 <code>exponentiate = TRUE</code>), NULL for not highlighting such coefficients
significance_labels	optional 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 <code>ggcoef_plot()</code>
table_stat	statistics to display in the table, use any column name returned by the tidier or "ci" for confidence intervals formatted according to <code>ci_pattern</code>
table_header	optional custom headers for the table
table_text_size	text size for the table
table_stat_label	optional 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
table_witdhs	[Deprecated] use <code>table_widths</code> instead
models	named list of models
type	a dodged plot, a faceted plot or multiple table plots?
data	a data frame containing data to be plotted, typically the output of <code>ggcoef_model()</code> , <code>ggcoef_compare()</code> or <code>ggcoef_multinom()</code> with the option <code>return_data = TRUE</code>

x, y	variables mapped to x and y axis
y_labeller	optional function to be applied on y labels (see examples)
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 <code>ggplot2::scale_colour_discrete()</code> and <code>ggplot2::discrete_scale()</code>
shape	optional variable name to be mapped to the shape aesthetic
shape_values	values of the different shapes to use in <code>ggplot2::scale_shape_manual()</code>
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 <code>exponentiate = TRUE</code>)?
vline_colour	colour of vertical line
dodged	should points be dodged (according to the colour aesthetic)?
dodged_width	width value for <code>ggplot2::position_dodge()</code>
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 <code>ggplot2::label_wrap_gen()</code> (see examples), more information in the documentation of <code>ggplot2::facet_grid()</code>
plot_title	an optional plot title
x_limits	optional limits for the x axis

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_dodged()`: a dodged variation of `ggcoef_model()` for multi groups models
- `ggcoef_faceted()`: a faceted variation of `ggcoef_model()` for multi groups models
- `ggcoef_compare()`: designed for displaying several models on the same plot.
- `ggcoef_plot()`: plot a tidy tibble of coefficients

Note

`ggcoef_compare(type = "table")` is not compatible with multi-components models.

See Also

`vignette("ggcoef_model")`

Examples

```
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 ",
  y_labeller = 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"
  )
)
# 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)
}

```

```

# multinomial model

```

```

mod <- nnet::multinom(grade ~ stage + trt + age, data = gtsummary::trial)
ggcoef_model(mod, exponentiate = TRUE)
ggcoef_table(mod, group_labels = c(II = "Stage 2 vs. 1"))
ggcoef_dodged(mod, exponentiate = TRUE)
ggcoef_faceted(mod, exponentiate = TRUE)

library(pscl)
data("bioChemists", package = "pscl")
mod <- zeroinfl(art ~ fem * mar | fem + mar, data = bioChemists)
ggcoef_model(mod)
ggcoef_table(mod)
ggcoef_dodged(mod)
ggcoef_faceted(
  mod,
  group_labels = c(conditional = "Count", zero_inflated = "Zero-inflated")
)

mod2 <- zeroinfl(art ~ fem + mar | 1, data = bioChemists)
ggcoef_table(mod2)
ggcoef_table(mod2, intercept = TRUE)

# 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")
ggcoef_compare(models, type = "table")

# 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)

```

Description**[Deprecated]****Usage**

```

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

```

Arguments

<code>model</code>	a regression model object
<code>type</code>	a dodged plot, a faceted plot or multiple table plots?
<code>component_col</code>	name of the component column
<code>component_label</code>	an optional named vector for labeling components
<code>tidy_fun</code>	(function) Option to specify a custom tidier function.
<code>tidy_args</code>	Additional arguments passed to <code>broom.helpers::tidy_plus_plus()</code> and to <code>tidy_fun</code>
<code>conf.int</code>	(logical) Should confidence intervals be computed? (see <code>broom::tidy()</code>)
<code>conf.level</code>	the confidence level to use for the confidence interval if <code>conf.int = TRUE</code> ; must be strictly greater than 0 and less than 1; defaults to 0.95, which corresponds to a 95 percent confidence interval
<code>exponentiate</code>	if TRUE a logarithmic scale will be used for x-axis
<code>variable_labels</code>	(formula-list-selector) A named list or a named vector of custom variable labels.
<code>term_labels</code>	(list or vector) A named list or a named vector of custom term labels.
<code>interaction_sep</code>	(string) Separator for interaction terms.
<code>categorical_terms_pattern</code>	(glue pattern) A glue pattern for labels of categorical terms with treatment or sum contrasts (see <code>model_list_terms_levels()</code>).

add_reference_rows	(logical) Should reference rows be added?
no_reference_row	(tidy-select) Variables for those no reference row should be added, when add_reference_rows = TRUE.
intercept	(logical) Should the intercept(s) be included?
include	(tidy-select) Variables to include. Default is everything(). See also all_continuous() , all_categorical() , all_dichotomous() and all_interaction() .
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	optional vector with custom labels for significance variable
return_data	if TRUE, will return the data.frame used for plotting instead of the 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	optional custom headers for the table
table_text_size	text size for the table
table_stat_label	optional named list of labeller functions for the displayed statistic (see examples)
ci_pattern	glue pattern for confidence intervals in the table
table_widths	[Deprecated] use table_widths instead
...	parameters passed to ggcoef_plot()
y.level_label	an optional named vector for labeling y.level (see examples)

Description

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.

Usage

```
gglikert(  
  data,  
  include = dplyr::everything(),  
  weights = NULL,  
  y = ".question",  
  variable_labels = NULL,  
  sort = c("none", "ascending", "descending"),  
  sort_method = c("prop", "prop_lower", "mean", "median"),  
  sort_prop_include_center = totals_include_center,  
  factor_to_sort = ".question",  
  exclude_fill_values = NULL,  
  cutoff = NULL,  
  data_fun = NULL,  
  add_labels = TRUE,  
  labels_size = 3.5,  
  labels_color = "auto",  
  labels_accuracy = 1,  
  labels_suffix = "%",  
  labels_hide_below = 0.05,  
  add_totals = TRUE,  
  totals_size = labels_size,  
  totals_color = "black",  
  totals_accuracy = labels_accuracy,  
  totals_suffix = labels_suffix,  
  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,  
  symmetric = FALSE  
)  
  
gglikert_data(  
  data,  
  include = dplyr::everything(),  
  weights = NULL,  
  variable_labels = NULL,  
  sort = c("none", "ascending", "descending"),  
  sort_method = c("prop", "prop_lower", "mean", "median"),  
  sort_prop_include_center = TRUE,  
  factor_to_sort = ".question",  
  exclude_fill_values = NULL,
```

```

    cutoff = NULL,
    data_fun = NULL
)

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

gglikert_side(
  data,
  include = dplyr::everything(),
  side_values,
  weights = NULL,
  y = ".question",
  variable_labels = NULL,
  sort = c("none", "ascending", "descending"),
  sort_method = c("prop", "prop_lower", "mean", "median"),
  sort_prop_include_center = totals_include_center,
  factor_to_sort = ".question",
  cutoff = NULL,
  data_fun = NULL,
  add_labels = TRUE,
  labels_size = 3.5,
  labels_color = "auto",
  labels_accuracy = 1,
  labels_suffix = "%",
  labels_hide_below = 0.05,
  add_totals = TRUE,

```

```

totals_size = labels_size,
totals_color = "black",
totals_accuracy = labels_accuracy,
totals_suffix = labels_suffix,
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,
symmetric = FALSE,
coord_ratio = NULL
)

```

Arguments

<code>data</code>	a data frame, a data frame extension (e.g. a tibble), or a survey design object
<code>include</code>	variables to include, accepts tidy-select syntax
<code>weights</code>	optional variable name of a weighting variable, accepts tidy-select syntax
<code>y</code>	name of the variable to be plotted on y axis (relevant when <code>.question</code> is mapped to "facets, see examples), accepts tidy-select syntax
<code>variable_labels</code>	a named list or a named vector of custom variable labels
<code>sort</code>	should the factor defined by <code>factor_to_sort</code> be sorted according to the answers (see <code>sort_method</code>)? One of "none" (default), "ascending" or "descending"
<code>sort_method</code>	method used to sort the variables: "prop" sort according to the proportion of answers higher than the centered level, "prop_lower" according to the proportion lower 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.
<code>sort_prop_include_center</code>	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?
<code>factor_to_sort</code>	name of the factor column to sort if <code>sort</code> is not equal to "none"; by default the list of questions passed to <code>include</code> ; should be one factor column of the tibble returned by <code>gglikert_data()</code> ; accepts tidy-select syntax
<code>exclude_fill_values</code>	Vector of values that should not be displayed (but still taken into account for computing proportions), see position_likert()
<code>cutoff</code>	number of categories to be displayed negatively (i.e. on the left of the x axis or the bottom of the y axis), could be a decimal value: 2 to display negatively the

	two first categories, 2.5 to display negatively the two first categories and half of the third, 2.2 to display negatively the two first categories and a fifth of the third (see examples). By default (NULL), it will be equal to the number of categories divided by 2, i.e. it will be centered.
data_fun	for advanced usage, custom function to be applied to the generated dataset at the end of <code>gglikert_data()</code>
add_labels	should percentage labels be added to the plot?
labels_size	size of the percentage labels
labels_color	color of the percentage labels ("auto" to use <code>hex_bw()</code> to determine a font color based on background color)
labels_accuracy	accuracy of the percentages, see <code>scales::label_percent()</code>
labels_suffix	text displayed after the percentages, see <code>scales::label_percent()</code>
labels_hide_below	if provided, values below will be masked, see <code>label_percent_abs()</code>
add_totals	should the total proportions of negative and positive answers be added to plot? You could also use "left" or "right" to display totals only on one side.
totals_size	size of the total proportions
totals_color	color of the total proportions
totals_accuracy	accuracy of the total proportions, see <code>scales::label_percent()</code>
totals_suffix	text displayed after the total proportions, see <code>scales::label_percent()</code>
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 <code>scales::label_wrap()</code>
reverse_likert	if TRUE, will reverse the default stacking order, see <code>position_likert()</code>
width	bar width, see <code>ggplot2::geom_bar()</code>
facet_rows, facet_cols	A set of variables or expressions quoted by <code>ggplot2::vars()</code> and defining faceting groups on the rows or columns dimension (see examples)
facet_label_wrap	number of characters per line for facet labels, see <code>ggplot2::label_wrap_gen()</code>
symmetric	should the x-axis be symmetric?
add_median_line	add a vertical line at 50%?
reverse_fill	if TRUE, will reverse the default stacking order, see <code>ggplot2::position_fill()</code>
side_values	values to be displayed on the side
coord_ratio	aspect ratio between x and y axis.

Details

You could use `gglikert_data()` to just produce the dataset to be plotted.

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

[Experimental] `gglikert_side()` allows to display certain values (e.g. don't knows) on the side.

Value

A `ggplot2` plot or a tibble.

See Also

`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)))

gglikert(df)

gglikert(df, include = q1:3) +
  scale_fill_likert(pal = scales::brewer_pal(palette = "PRGn"))

gglikert(df, sort = "ascending")

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

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

gglikert(df, reverse_likert = TRUE)
```

```

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

gglikert(
  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,
  labels_suffix = " %"
)

gglikert(df, exclude_fill_values = "Neither agree nor disagree")

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 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))

# Custom function to be applied on data
f <- function(d) {
  d$.question <- forcats::fct_relevel(d$.question, "q5", "q2")
  d
}
gglikert(df, include = q1:q6, data_fun = f)

# Custom center
gglikert(df, cutoff = 2)

```

```

gglikert(df, cutoff = 1)

gglikert(df, cutoff = 1, symmetric = TRUE)

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))

gglikert_side(df, side_values = "Neither agree nor disagree")

gglikert_side(
  df,
  side_values = "Neither agree nor disagree",
  coord_ratio = 1/3
)

gglikert_side(
  df,
  side_values = "Neither agree nor disagree",
  cutoff = 0,
  add_totals = FALSE
)

```

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 <code>survey::svydesign()</code>
mapping	Default list of aesthetic mappings to use for plot, to be created with <code>ggplot2::aes()</code> .
...	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

```
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")
```

hex_bw	<i>Identify a suitable font color (black or white) given a background HEX color</i>
--------	---

Description

You could use `auto_contrast` as a shortcut of `aes(colour = after_scale(hex_bw(.data$fill)))`. You should use `!!!` to inject it within `ggplot2::aes()` (see examples).

`hex_bw_threshold()` is a variation of `hex_bw()`. For values below threshold, black ("`#000000`") will always be returned, regardless of `hex_code`.

Usage

```
hex_bw(hex_code)

hex_bw_threshold(hex_code, values, threshold)

auto_contrast
```

Arguments

hex_code	Background color in hex-format.
values	Values to be compared.
threshold	Threshold.

Value

Either black or white, in hex-format

Source

Adapted from saros for `hex_code()` and from https://github.com/teunbrand/ggplot_tricks?tab=readme-ov-file#text-contrast for `auto_contrast`.

Examples

```
hex_bw("#0dadfd")

library(ggplot2)
ggplot(diamonds) +
  aes(x = cut, fill = color, label = after_stat(count)) +
  geom_bar() +
  geom_text(
    mapping = aes(color = after_scale(hex_bw(.data$fill))),
    position = position_stack(.5),
    stat = "count",
    size = 2
  )

ggplot(diamonds) +
  aes(x = cut, fill = color, label = after_stat(count)) +
  geom_bar() +
  geom_text(
    mapping = auto_contrast,
    position = position_stack(.5),
    stat = "count",
    size = 2
  )

ggplot(diamonds) +
  aes(x = cut, fill = color, label = after_stat(count), !!!auto_contrast) +
  geom_bar() +
  geom_text(
    mapping = auto_contrast,
    position = position_stack(.5),
    stat = "count",
    size = 2
  )
```

label_number_abs	<i>Label absolute values</i>
------------------	------------------------------

Description

Label absolute values

Usage

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

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)

Value

A "labelling" function, i.e. a function that takes a vector and returns a character vector of same length giving a label for each input value.

See Also

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

Examples

```
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)
```

pal_extender

Extend a discrete colour palette

Description

If the palette returns less colours than requested, the list of colours will be expanded using `scales::pal_gradient_n()`. To be used with a sequential or diverging palette. Not relevant for qualitative palettes.

Usage

```
pal_extender(pal = scales::brewer_pal(palette = "BrBG"))

scale_fill_extended(
  name = waiver(),
  ...,
  pal = scales::brewer_pal(palette = "BrBG"),
  aesthetics = "fill"
```

```

)

scale_colour_extended(
  name = waiver(),
  ...,
  pal = scales::brewer_pal(palette = "BrBG"),
  aesthetics = "colour"
)

```

Arguments

pal	A palette function, such as returned by scales::brewer_pal , taking a number of colours as entry and returning a list of colours.
name	The name of the scale. Used as the axis or legend title. If <code>waiver()</code> , the default, the name of the scale is taken from the first mapping used for that aesthetic. If <code>NULL</code> , the legend title will be omitted.
...	Other arguments passed on to <code>discrete_scale()</code> 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 <code>aesthetics = c("colour", "fill")</code> .

Value

A palette function.

Examples

```

pal <- scales::pal_brewer(palette = "PiYG")
scales::show_col(pal(16))
scales::show_col(pal_extender(pal)(16))

```

position_likert

Stack objects on top of each another and center them around 0

Description

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

Usage

```
position_likert(
  vjust = 1,
  reverse = FALSE,
  exclude_fill_values = NULL,
  cutoff = NULL
)

position_diverging(
  vjust = 1,
  reverse = FALSE,
  exclude_fill_values = NULL,
  cutoff = NULL
)
```

Arguments

<code>vjust</code>	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.
<code>reverse</code>	If TRUE, will reverse the default stacking order. This is useful if you're rotating both the plot and legend.
<code>exclude_fill_values</code>	Vector of values from the variable associated with the fill aesthetic that should not be displayed (but still taken into account for computing proportions)
<code>cutoff</code>	number of categories to be displayed negatively (i.e. on the left of the x axis or the bottom of the y axis), could be a decimal value: 2 to display negatively the two first categories, 2.5 to display negatively the two first categories and half of the third, 2.2 to display negatively the two first categories and a fifth of the third (see examples). By default (NULL), it will be equal to the number of categories divided by 2, i.e. it will be centered.

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

```
library(ggplot2)

ggplot(diamonds) +
  aes(y = clarity, fill = cut) +
  geom_bar(position = "fill") +
```

```

    scale_x_continuous(label = scales::label_percent()) +
    xlab("proportion")

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

ggplot(diamonds) +
  aes(y = clarity, fill = cut) +
  geom_bar(position = "stack") +
  scale_fill_likert(pal = scales::brewer_pal(palette = "PiYG"))

ggplot(diamonds) +
  aes(y = clarity, fill = cut) +
  geom_bar(position = "diverging") +
  scale_x_continuous(label = label_number_abs()) +
  scale_fill_likert()

# 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_likert() +
  xlab("proportion")

# Custom center -----
ggplot(diamonds) +
  aes(y = clarity, fill = cut) +
  geom_bar(position = position_likert(cutoff = 1)) +
  scale_x_continuous(label = label_percent_abs()) +
  scale_fill_likert(cutoff = 1) +
  xlab("proportion")

ggplot(diamonds) +
  aes(y = clarity, fill = cut) +
  geom_bar(position = position_likert(cutoff = 3.75)) +
  scale_x_continuous(label = label_percent_abs()) +
  scale_fill_likert(cutoff = 3.75) +
  xlab("proportion")

# Missing items -----
# example with a level not being observed for a specific value of y
d <- diamonds
d <- d[!(d$cut == "Premium" & d$clarity == "I1"), ]
d <- d[!(d$cut %in% c("Fair", "Good") & d$clarity == "SI2"), ]

```

```

# by default, the two lowest bar are not properly centered
ggplot(d) +
  aes(y = clarity, fill = cut) +
  geom_bar(position = "likert") +
  scale_fill_likert()

# 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_likert()

# 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_likert() +
  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_likert() +
  xlab("proportion")

```

round_any

Round to multiple of any number.

Description

Round to multiple of any number.

Usage

```
round_any(x, accuracy, f = round)
```

Arguments

x	numeric or date-time (POSIXct) vector to round
accuracy	number to round to; for POSIXct objects, a number of seconds
f	rounding function: floor , ceiling or round

Source

adapted from plyr

Examples

```
round_any(1.865, accuracy = .25)
```

scale_fill_likert	<i>Colour scale for Likert-type plots</i>
-------------------	---

Description

This scale is similar to other diverging discrete colour scales, but allows to change the "center" of the scale using cutoff argument, as used by [position_likert\(\)](#).

Usage

```
scale_fill_likert(
  name = waiver(),
  ...,
  pal = scales::brewer_pal(palette = "BrBG"),
  cutoff = NULL,
  aesthetics = "fill"
)

likert_pal(pal = scales::brewer_pal(palette = "BrBG"), cutoff = NULL)
```

Arguments

name	The name of the scale. Used as the axis or legend title. If <code>waiver()</code> , the default, the name of the scale is taken from the first mapping used for that aesthetic. If <code>NULL</code> , the legend title will be omitted.
...	Other arguments passed on to <code>discrete_scale()</code> to control name, limits, breaks, labels and so forth.
pal	A palette function taking a number of colours as entry and returning a list of colours (see examples), ideally a diverging palette

cutoff	Number of categories displayed negatively (see position_likert()) and therefore changing the center of the colour scale (see examples).
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").

Examples

```
library(ggplot2)
ggplot(diamonds) +
  aes(y = clarity, fill = cut) +
  geom_bar(position = "likert") +
  scale_x_continuous(label = label_percent_abs()) +
  xlab("proportion")

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

ggplot(diamonds) +
  aes(y = clarity, fill = cut) +
  geom_bar(position = position_likert(cutoff = 1)) +
  scale_x_continuous(label = label_percent_abs()) +
  xlab("proportion") +
  scale_fill_likert(cutoff = 1)
```

signif_stars	<i>Significance Stars</i>
--------------	---------------------------

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

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

stat_cross

Compute cross-tabulation statistics

Description

Computes statistics of a 2-dimensional matrix using [broom::augment.htest](#).

Usage

```
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 <code>inherit.aes = TRUE</code> (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 <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to ggplot(). A <code>data.frame</code>, 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 <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).

geom	Override the default connection with <code>ggplot2::geom_point()</code> .
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as <code>position_jitter()</code>. This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <code>position_jitter()</code>, give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to <code>layer()</code>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can <i>not</i> be passed through ... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an Aesthetics section that lists the available options. The 'required' aesthetics cannot be passed on to the <code>params</code>. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data. • When constructing a layer using a <code>stat_*()</code> function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is <code>stat_density(geom = "area", outline.type = "both")</code>. The geom's documentation lists which parameters it can accept. • Inversely, when constructing a layer using a <code>geom_*()</code> function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is <code>geom_area(stat = "density", adjust = 0.5)</code>. The stat's documentation lists which parameters it can accept. • The <code>key_glyph</code> argument of <code>layer()</code> may also be passed on through ... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.
na.rm	If TRUE, the default, missing values are removed with a warning. If FALSE, 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. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
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. <code>annotation_borders()</code> .
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
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

```
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)
  ) +
  geom_text(stat = "cross")

# Row proportions with standardized residuals
ggplot(d) +
  aes(
    x = Class, y = Survived, weight = Freq,
    label = scales::percent(after_stat(row.prop)),
    size = NULL, fill = after_stat(std.resid)
  ) +
  stat_cross(shape = 22, size = 30) +
  geom_text(stat = "cross") +
  scale_fill_steps2(breaks = c(-3, -2, 2, 3), show.limits = TRUE) +
  facet_grid(Sex ~ .) +
  labs(fill = "Standardized residuals") +
  theme_minimal()
```

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). If the **by** aesthetic is not specified, denominators will be determined according to the `default_by` argument.

Usage

```
stat_prop(
  mapping = NULL,
  data = NULL,
  geom = "bar",
  position = "fill",
```

```

...,
na.rm = FALSE,
orientation = NA,
show.legend = NA,
inherit.aes = TRUE,
complete = NULL,
default_by = "total"
)

```

Arguments

mapping	Set of aesthetic mappings created by aes() . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to ggplot(). A <code>data.frame</code>, 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 <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).
geom	Override the default connection with ggplot2::geom_bar() .
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to layer()'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can <i>not</i> be passed through Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an Aesthetics section that lists the available options. The 'required' aesthetics cannot be passed on to the <code>params</code>. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.

	• When constructing a layer using a <code>stat_*</code>() function, the <code>...</code> argument can be used to pass on parameters to the geom part of the layer. An example of this is <code>stat_density(geom = "area", outline.type = "both")</code>. The geom's documentation lists which parameters it can accept. • Inversely, when constructing a layer using a <code>geom_*</code>() function, the <code>...</code> argument can be used to pass on parameters to the stat part of the layer. An example of this is <code>geom_area(stat = "density", adjust = 0.5)</code>. The stat's documentation lists which parameters it can accept. • The <code>key_glyph</code> argument of <code>layer()</code> may also be passed on through <code>...</code>. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.
<code>na.rm</code>	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
<code>orientation</code>	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 <i>Orientation</i> section for more detail.
<code>show.legend</code>	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. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
<code>inherit.aes</code>	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. <code>annotation_borders()</code> .
<code>complete</code>	Name (character) of an aesthetic for those statistics should be completed for unobserved values (see example).
<code>default_by</code>	If the by aesthetic is not available, name of another aesthetic that will be used to determine the denominators (e.g. "fill"), or NULL or "total" to compute proportions of the total. To be noted, <code>default_by = "x"</code> works both for vertical and horizontal bars.

Value

A ggplot2 plot with the added statistic.

Aesthetics

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

- **x or y**
- by
- weight

Computed variables

after_stat(count) number of points in bin
 after_stat(denominator) denominator for the proportions
 after_stat(prop) computed proportion, i.e. after_stat(count)/after_stat(denominator)

See Also

vignette("stat_prop"), `ggplot2::stat_count()`. For an alternative approach, see <https://github.com/tidyverse/ggplot2/issues/5505#issuecomment-1791324008>.

Examples

```
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)

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"
  )

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
d <- diamonds |>
  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	<i>Compute weighted y mean</i>
--------------------	--------------------------------

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 <code>inherit.aes = TRUE</code> (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 | <p>The data to be displayed in this layer. There are three options:</p> <p>If <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to ggplot().</p> <p>A <code>data.frame</code>, 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.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p> |
| geom | Override the default connection with ggplot2::geom_point() . |
| position | <p>A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:</p> <ul style="list-style-type: none"> • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. |

- A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use `position_jitter()`, give the position as "jitter".
- For more information and other ways to specify the position, see the [layer position](#) documentation.

...

Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through ... Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through ... This can be one of the functions described as [key glyphs](#), to change the display of the layer in the legend.

<code>na.rm</code>	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
<code>orientation</code>	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 <i>Orientation</i> section for more detail.
<code>show.legend</code>	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. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
<code>inherit.aes</code>	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. <code>annotation_borders()</code> .

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(ggplot2)

# taking into account some weights
d <- as.data.frame(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")
```

symmetric_limits	<i>Symmetric limits</i>
------------------	-------------------------

Description

Expand scale limits to make them symmetric around zero. Can be passed as argument to parameter `limits` of continuous scales from packages `{ggplot2}` or `{scales}`. Can be also used to obtain an enclosing symmetric range for numeric vectors.

Usage

```
symmetric_limits(x)
```

Arguments

<code>x</code>	a vector of numeric values, possibly a range, from which to compute enclosing range
----------------	---

Value

A numeric vector of length two with the new limits, which are always such that the absolute value of upper and lower limits is the same.

Source

Adapted from the homonym function in `{ggpmisc}`

Examples

```
library(ggplot2)  
  
ggplot(iris) +  
  aes(x = Sepal.Length - 5, y = Sepal.Width - 3, colour = Species) +  
  geom_vline(xintercept = 0) +  
  geom_hline(yintercept = 0) +  
  geom_point()  
  
last_plot() +  
  scale_x_continuous(limits = symmetric_limits) +  
  scale_y_continuous(limits = symmetric_limits)
```

weighted.median	<i>Weighted Median and Quantiles</i>
-----------------	--------------------------------------

Description

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

Usage

```
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 <code>stats::quantile()</code> . The only valid choices are <code>type=1</code> , <code>2</code> or <code>4</code> . 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 `type = 1`, the next largest value is returned (this is the right-continuous inverse of the left-continuous cumulative distribution function);
- if `type = 2`, the average of the two surrounding values is returned (the average of the right-continuous and left-continuous inverses);
- if `type = 4`, linear interpolation is performed.

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

Value

A numeric vector.

Source

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

Examples

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

weighted.sum	<i>Weighted Sum</i>
--------------	---------------------

Description

Weighted Sum

Usage

```
weighted.sum(x, w, na.rm = TRUE)
```

Arguments

x	a numeric vector of values
w	a numeric vector of weights
na.rm	a logical indicating whether to ignore NA values

Value

A numeric vector.

Examples

```
x <- 1:20
w <- runif(20)
weighted.sum(x, w)
```

Index

`aes()`, 4, 10, 13, 47, 51, 54
`all_categorical()`, 21, 30
`all_continuous()`, 21, 30
`all_dichotomous()`, 21, 30
`all_interaction()`, 21, 30
`annotation_borders()`, 5, 14, 48, 52, 55
`augment_chisq_add_phi`, 2
`augment_chisq_add_phi()`, 49
`auto_contrast` (`hex_bw`), 38

`broom.helpers::tidy_plus_plus()`, 17, 21, 29
`broom::augment.htest`, 47
`broom::tidy()`, 21, 29

`ceiling`, 45
`compute_cascade` (`ggcascade`), 15

`dplyr::pick()`, 16

`emmeans::contrast()`, 22
`emmeans::emmeans()`, 22

`floor`, 45
`fortify()`, 4, 10, 13, 47, 51, 54

`geom_bar_connector` (`geom_connector`), 3
`geom_bar_connector()`, 10, 11
`geom_connector`, 3
`geom_diverging`, 7
`geom_diverging_text` (`geom_diverging`), 7
`geom_likert` (`geom_diverging`), 7
`geom_likert_text` (`geom_diverging`), 7
`geom_prop_bar`, 9
`geom_prop_connector` (`geom_prop_bar`), 9
`geom_prop_text` (`geom_prop_bar`), 9
`geom_pyramid` (`geom_diverging`), 7
`geom_pyramid_text` (`geom_diverging`), 7
`geom_stripped_cols`
 (`geom_stripped_rows`), 12
`geom_stripped_rows`, 12

`GeomConnector` (`geom_connector`), 3
`ggcascade`, 15
`ggcoef_compare` (`ggcoef_model`), 17
`ggcoef_dodged` (`ggcoef_model`), 17
`ggcoef_faceted` (`ggcoef_model`), 17
`ggcoef_model`, 17
`ggcoef_model()`, 24
`ggcoef_multicomponents`, 27
`ggcoef_multinom`
 (`ggcoef_multicomponents`), 27
`ggcoef_plot` (`ggcoef_model`), 17
`ggcoef_plot()`, 22, 30
`ggcoef_table` (`ggcoef_model`), 17
`gglikert`, 30
`gglikert_data` (`gglikert`), 30
`gglikert_side` (`gglikert`), 30
`gglikert_stacked` (`gglikert`), 30
`ggplot()`, 4, 10, 13, 47, 51, 54
`ggplot2::aes()`, 37, 38
`ggplot2::discrete_scale()`, 23
`ggplot2::facet_grid()`, 23
`ggplot2::geom_bar()`, 7, 8, 10, 11, 34, 51
`ggplot2::geom_point()`, 48, 54
`ggplot2::geom_smooth()`, 37
`ggplot2::geom_step()`, 3
`ggplot2::geom_text()`, 7, 10, 11, 16
`ggplot2::label_wrap_gen()`, 23, 34
`ggplot2::position_dodge()`, 23
`ggplot2::position_fill()`, 34, 42
`ggplot2::position_stack()`, 42
`ggplot2::scale_colour_discrete()`, 23
`ggplot2::scale_shape_manual()`, 23
`ggplot2::stat_count()`, 50, 53
`ggplot2::vars()`, 34
`ggsurvey`, 37
`glue` pattern, 21, 29

`hex_bw`, 38
`hex_bw_threshold` (`hex_bw`), 38

key glyphs, [5](#), [14](#), [48](#), [52](#), [55](#)

label_number_abs, [39](#)

label_percent_abs (label_number_abs), [39](#)

label_percent_abs(), [34](#)

labelled::var_label(), [35](#)

layer position, [5](#), [11](#), [13](#), [48](#), [51](#), [55](#)

layer stat, [4](#), [13](#)

layer(), [5](#), [13](#), [14](#), [48](#), [51](#), [52](#), [55](#)

likert_pal (scale_fill_likert), [45](#)

model_list_terms_levels(), [21](#), [29](#)

pal_extender, [40](#)

plot_cascade (ggcascade), [15](#)

position_diverging (position_likert), [41](#)

position_diverging(), [8](#)

position_likert, [41](#)

position_likert(), [8](#), [33–35](#), [45](#), [46](#)

PositionDiverging (position_likert), [41](#)

PositionLikert (position_likert), [41](#)

round, [45](#)

round_any, [44](#)

scale_colour_extended (pal_extender), [40](#)

scale_fill_extended (pal_extender), [40](#)

scale_fill_likert, [45](#)

scales::brewer_pal, [41](#)

scales::label_number(), [40](#)

scales::label_percent(), [34](#), [40](#)

scales::label_wrap(), [34](#)

scales::pal_gradient_n(), [40](#)

signif_stars, [46](#)

stat_cross, [47](#)

stat_cross(), [3](#)

stat_prop, [50](#)

stat_prop(), [8–10](#), [35](#)

stat_weighted_mean, [54](#)

StatCross (stat_cross), [47](#)

StatProp (stat_prop), [50](#)

stats::chisq.test(), [2](#)

StatWeightedMean (stat_weighted_mean),
[54](#)

survey::svydesign(), [37](#)

symmetric_limits, [57](#)

tidy-select, [33](#)

tidy_add_pairwise_contrasts(), [22](#)

weighted.median, [58](#)

weighted.quantile (weighted.median), [58](#)

weighted.sum, [59](#)