Package ‘ggstats’

November 24, 2022

Type Package

Title Extension to 'ggplot2' for Plotting Stats

Version 0.1.1

Description Provides suite of functions to plot regression model coefficients ("forest plots"). The suite also includes new statistics to compute proportions, weighted mean and cross-tabulation statistics, as well as new geometries to add alternative background color to a plot.

License GPL-3

URL https://larmarange.github.io/ggstats/

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

Imports broom.helpers, cli, dplyr, forcats, ggplot2, lifecycle, magrittr, scales, stats, tidyr

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

Encoding UTF-8

RoxygenNote 7.2.1

Config/testthat/edition 3

Language en-US

VignetteBuilder knitr

NeedsCompilation no

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

Maintainer Joseph Larmarange <joseph@larmarange.net>

Repository CRAN

Date/Publication 2022-11-24 00:20:08 UTC

R topics documented:

  augment_chisq_add_phi ......................................................... 2
  geom_stripped_rows ............................................................. 3
augment_chisq_add_phi

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))
Alternating Background Color

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

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

```r
gem_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()` or `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
- **data**: The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. 


geom_stripped_rows

A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.

A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).

stat
The statistical transformation to use on the data for this layer, as a string.

position
Position adjustment, either as a string, or the result of a call to a position adjustment function.

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

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

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

xlim, xto limitation of the strips along the x-axis
width width of the strips
ylim, yto limitation of the strips along the y-axis
nudge_x, nudge_y 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() +
ggcoef_model

- `theme_light()`
- `p + geom_stripped_rows()`
- `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)`

---

**ggcoef_model**

Plot model coefficients

---

**Description**

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

**Usage**

```r
ggcoef_model(
  model,
  tidy_fun = broom.helpers::tidy_with_broom_or_parameters,
  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,
  show_p_values = TRUE,
  signif_stars = TRUE,
  return_data = FALSE,
...)
```
ggcoef_compare(
  models,
  type = c("dodged", "faceted"),
  tidy_fun = broom.helpers::tidy_with_broom_or_parameters,
  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,
  ...
)

ggcoef_multinom(
  model,
  type = c("dodged", "faceted"),
  y.level_label = NULL,
  tidy_fun = broom.helpers::tidy_with_broom_or_parameters,
  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,
  show_p_values = TRUE,
  signif_stars = TRUE,
  return_data = FALSE,
  ...
)

ggcoef_plot(
  data,
  x = "estimate",
  ...
ggcof_model

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

**Arguments**

- **model**: a regression model object
- **tidy_fun**: option to specify a custom tidier function
- **conf.int**: should confidence intervals be computed? (see `broom::tidy()`)
- **conf.level**: the confidence level to use for the confidence interval if `conf.int = TRUE`; must be strictly greater than 0 and less than 1; defaults to 0.95, which corresponds to a 95 percent confidence interval
- **exponentiate**: if TRUE a logarithmic scale will be used for x-axis
- **variable_labels**: a named list or a named vector of custom variable labels
- **term_labels**: a named list or a named vector of custom term labels
- **interaction_sep**: separator for interaction terms
- **categorical_terms_pattern**: a glue pattern for labels of categorical terms with treatment or sum contrasts (see `model_list_terms_levels()`)
- **add_reference_rows**: should reference rows be added?
no_reference_row
variables (accepts tidyselect notation) for those no reference row should be
added, when add_reference_rows = TRUE

intercept
should the intercept(s) be included?

include
variables to include. Accepts tidyselect syntax. Use ~ to remove a variable.
Default is everything(). See also all_continuous(), all_categorical(),
all_dichotomous() and all_interaction()

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

show_p_values
if TRUE, add p-value to labels

signif_stars
if TRUE, add significant stars to labels

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

... parameters passed to ggcoef_plot()

models
named list of models

type
a dodged plot or a faceted plot?

y.level_label
an optional named vector for labeling y.level (see examples)

data
a data frame containing data to be plotted, typically the output of ggcoef_model(),
ggcoef_compare() or ggcoef_multinom() with the option return_data = TRUE

x, y
variables mapped to x and y axis

point_size
size of the points

point_stroke
thickness of the points

point_fill
fill colour for the points

colour
optional variable name to be mapped to colour aesthetic

colour_guide
should colour guide be displayed in the legend?

colour_lab
label of the colour aesthetic in the legend

colour_labels
labels argument passed to ggplot2::scale_colour_discrete() and ggplot2::discrete_scale()

shape
optional variable name to be mapped to the shape aesthetic

shape_values
values of the different shapes to use in ggplot2::scale_shape_manual()

shape_guide
should shape guide be displayed in the legend?

shape_lab
label of the shape aesthetic in the legend

derrorbar
should error bars be plotted?

errorbar_height
height of error bars

derrorbar_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
**ggcoef_model**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strips_even</td>
<td>color of the even rows</td>
</tr>
<tr>
<td>vline</td>
<td>should a vertical line be drawn at 0 (or 1 if exponentiate = TRUE)?</td>
</tr>
<tr>
<td>vline_colour</td>
<td>colour of vertical line</td>
</tr>
<tr>
<td>dodged</td>
<td>should points be dodged (according to the colour aesthetic)?</td>
</tr>
<tr>
<td>dodged_width</td>
<td>width value for <code>ggplot2::position_dodge()</code></td>
</tr>
<tr>
<td>facet_row</td>
<td>variable name to be used for row facets</td>
</tr>
<tr>
<td>facet_col</td>
<td>optional variable name to be used for column facets</td>
</tr>
<tr>
<td>facet_labeller</td>
<td>labeller function to be used for labeling facets; if labels are too long,</td>
</tr>
<tr>
<td></td>
<td>you can use <code>ggplot2::label_wrap_gen()</code> (see examples), more information in</td>
</tr>
<tr>
<td></td>
<td>the documentation of <code>ggplot2::facet_grid()</code></td>
</tr>
</tbody>
</table>

**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_compare()`: designed for displaying several models on the same plot.
- `ggcoef_multinom()`: a variation of `ggcoef_model()` adapted to multinomial logistic regressions performed with `nnet::multinom()`.
- `ggcoef_plot()`: plot a tidy tibble of coefficients

**Examples**

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

# 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)

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

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

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

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

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

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

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

```r

# you can provide custom variable labels with 'variable_labels'

```
"Simplified model" = mod2,
"With interaction" = mod3
)

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

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

# specific function for nnet::multinom models
mod <- nnet::multinom(Species ~ ., data = iris)
ggcoef_multinom(mod, exponentiate = TRUE)
ggcoef_multinom(mod, type = "faceted")
ggcoef_multinom(
  mod,
  type = "faceted",
  y.level_label = c("versicolor" = "versicolor\n(ref: setosa)"
)

---

ggsurvey

Easy ggplot2 with survey objects

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

Usage

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

Arguments

design
  A survey design object, usually created with survey::svydesign()
mapping
  Default list of aesthetic mappings to use for plot, to be created with ggplot2::aes().
...
  Other arguments passed on to methods. Not currently used.

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

**Value**

A ggplot2 plot.

**Examples**

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

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

---

**signif_stars**

**Significance Stars**

**Description**

Calculate significance stars

**Usage**

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

**Arguments**

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

**Value**

Character vector containing the appropriate number of stars for each x value.
Author(s)

Joseph Larmarange

Examples

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

---

**stat_cross**

*Compute cross-tabulation statistics*

**Description**

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

**Usage**

```r
stat_cross(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...
)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mapping</code></td>
<td>Set of aesthetic mappings created by <code>aes()</code> or <code>aes()</code>. If specified and <code>inherit.aes</code> = <code>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.</td>
</tr>
<tr>
<td><code>data</code></td>
<td>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 <code>formula</code> (e.g. <code>~ head(.x, 10)</code>).</td>
</tr>
<tr>
<td><code>geom</code></td>
<td>Override the default connection with <code>ggplot2::geom_point()</code>.</td>
</tr>
</tbody>
</table>
Position adjustment, either as a string, or the result of a call to a position adjust-
ment function.

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

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

If TRUE, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

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

If TRUE, cells with no observations are kept.

A ggplot2 plot with the added statistic.

`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()`

Examples

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

# plot number of observations
ggplot(d) +
```
Compute proportions according to custom denominator
Description

`stat_prop()` is a variation of `ggplot2::stat_count()` allowing to compute custom proportions according to the by aesthetic defining the denominator (i.e. all proportions for a same value of by will sum to 1). The by aesthetic should be a factor.

Usage

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

Arguments

- **mapping**: Set of aesthetic mappings created by `aes()` or `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
- **data**: The data to be displayed in this layer. There are three options:
  - If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
  - A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
  - A function will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x, 10)`).
- **geom**: Override the default connection with `ggplot2::geom_bar()`.
- **position**: Position adjustment, either as a string, or the result of a call to a position adjustment function.
- **width**: Bar width. By default, set to 90% of the resolution of the data.
- **na.rm**: If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.
- **orientation**: The orientation of the layer. The default (NA) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be given explicitly by setting orientation to either "x" or "y". See the Orientation section for more detail.
show.legend  logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

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

Value

A ggplot2 plot with the added statistic.

Aesthetics

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

- x or y
- by (this aesthetic should be a factor)
- group
- weight

Computed variables

- count number of points in bin
- prop computed proportion

See Also

ggplot2::stat_count()

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)
    )
}

stat_weighted_mean  Compute weighted y mean

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

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

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

- **geom**: Override the default connection with `ggplot2::geom_point()`.
- **position**: Position adjustment, either as a string, or the result of a call to a position adjustment function.
- **...**: Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.
- **na.rm**: If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
- **orientation**: The orientation of the layer. The default (NA) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be given explicitly by setting orientation to either "x" or "y". See the Orientation section for more detail.
- **show.legend**: logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
- **inherit.aes**: If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders()`.

**Value**

A `ggplot2` plot with the added statistic.

**Computed variables**

- **y**: weighted y (numerator / denominator)
- **numerator**: numerator
- **denominator**: denominator

**Examples**

```r
e e 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")
```
```r
# 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)
}
```

```r
# taking into account some weights
if (requireNamespace("scales")) {
  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")
}
```
Index

* datasets
  - stat_cross, 14
  - stat_prop, 16
  - stat_weighted_mean, 19

  aes(), 3, 14, 17, 19
  aes_(), 3, 14, 17, 19
  all_categorical(), 8
  all_continuous(), 8
  all_dichotomous(), 8
  all_interaction(), 8
  augment_chisq_add_phi, 2
  augment_chisq_add_phi(), 15

  borders(), 4, 15, 18, 20
  broom.helpers::tidy_plus_plus(), 5
  broom::augment.htest, 14
  broom::tidy(), 7

  fortyy(), 4, 14, 17, 19

  geom_stripped_cols
    (geom_stripped_rows), 3
  geom_stripped_rows, 3
  ggcoef_compare (ggcoef_model), 5
  ggcoef_model, 5
  ggcoef_model(), 9
  ggcoef_multinom (ggcoef_model), 5
  ggcoef_plot (ggcoef_model), 5
  ggcoef_plot(), 8
  ggplot(), 3, 14, 17, 19
  ggplot2::aes(), 12
  ggplot2::discrete_scale(), 8
  ggplot2::facet_grid(), 9
  ggplot2::geom_bar(), 17
  ggplot2::geom_point(), 14, 20
  ggplot2::geom_smooth(), 12
  ggplot2::label_wrap_gen(), 9
  ggplot2::position_dodge(), 9
  ggplot2::scale_colour_discrete(), 8
  ggplot2::scale_shape_manual(), 8
  ggplot2::stat_count(), 17, 18
  ggssurvey, 12
  glue pattern, 7

  layer(), 4, 15, 17, 20
  model_list_terms_levels(), 7
  nnet::multinom(), 9
  signif_stars, 13
  stat_cross, 14
  stat_cross(), 2
  stat_prop, 16
  stat_weighted_mean, 19
  StatCross (stat_cross), 14
  StatProp (stat_prop), 16
  stats::chisq.test(), 2
  StatWeightedMean (stat_weighted_mean), 19
  survey::svydesign(), 12

  tidyselect, 8