Package ‘ggpmisc’

August 15, 2021

Type Package

Title Miscellaneous Extensions to 'ggplot2'

Version 0.4.2-1

Date 2021-08-14

Maintainer Pedro J. Aphalo <pedro.aphalo@helsinki.fi>

Description Extensions to 'ggplot2' respecting the grammar of graphics paradigm. Statistics: locate and tag peaks and valleys; label plot with the equation of a polynomial fitted with lm() or other types of models; labels with P-value, R^2 or adjusted R^2 or information criteria for fitted models; label with ANOVA table for fitted models; label with summary for fitted models. Model fit classes for which suitable methods are provided by package 'broom' and 'broom.mixed' are supported. Scales and stats to build volcano and quadrant plots based on outcomes, fold changes, p-values and false discovery rates.

License GPL (>= 2)

LazyData TRUE

LazyLoad TRUE

ByteCompile TRUE

Depends R (>= 3.6.0), ggpp (>= 0.4.0)

Imports grid, stats, ggplot2 (>= 3.3.1), scales (>= 1.1.1), rlang (>= 0.4.7), generics (>= 0.1.0), MASS (>= 7.3-51.6), polynom (>= 1.4-0), quantreg (>= 5.85), splut2R (>= 1.2-2), tibble (>= 3.0.3), plyr (>= 1.8.6), dplyr (>= 1.0.2), lubridate (>= 1.7.9)

Suggests knitr (>= 1.29), rmarkdown (>= 2.3), ggrepel (>= 0.9.1), broom (>= 0.7.3), broom.mixed (>= 0.2.6), nlme (>= 3.1-148), gginnards(>= 0.0.3)

URL https://docs.r4photobiology.info/ggpmisc/,
https://github.com/aphalo/ggpmisc

BugReports https://github.com/aphalo/ggpmisc/issues

Encoding UTF-8
RoxygenNote 7.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Pedro J. Aphalo [aut, cre] (<https://orcid.org/0000-0003-3385-972X>),
    Kamil Slowikowski [ctb] (<https://orcid.org/0000-0002-2843-6370>),
    Samer Mouksassi [ctb] (<https://orcid.org/0000-0002-7152-6654>)

Repository CRAN

Date/Publication 2021-08-15 09:00:02 UTC

R topics documented:

- ggpmisc-package .................................................. 3
- Moved ................................................................. 5
- outcome2factor .................................................... 5
- quadrant_example.df ................................................. 6
- scale_colour_outcome ............................................ 7
- scale_shape_outcome ............................................. 9
- scale_x_logFC ..................................................... 10
- scale_y_Pvalue ................................................... 13
- stat_fit_augment .................................................. 15
- stat_fit_deviations ............................................... 19
- stat_fit_glance ................................................... 22
- stat_fit_residuals ................................................ 26
- stat_fit_tb ......................................................... 30
- stat_fit_tidy ...................................................... 35
- stat_peaks ......................................................... 39
- stat_poly_eq ....................................................... 43
- stat_poly_line .................................................... 50
- stat_quant_band .................................................. 53
- stat_quant_eq ..................................................... 56
- stat_quant_line ................................................... 64
- swap_xy ............................................................. 67
- symmetric_limits .................................................. 68
- volcano_example.df .............................................. 69
- xy_outcomes2factor ............................................... 69

Index 71
ggpmisc-package

GGPMISC: Miscellaneous Extensions to 'ggplot2'

Description

Extensions to 'ggplot2' respecting the grammar of graphics paradigm. Statistics: locate and tag peaks and valleys; label plot with the equation of a polynomial fitted with lm() or other types of models; labels with P-value, R^2 or adjusted R^2 or information criteria for fitted models; label with ANOVA table for fitted models; label with summary for fitted models. Model fit classes for which suitable methods are provided by package 'broom' and 'broom.mixed' are supported. Scales and stats to build volcano and quadrant plots based on outcomes, fold changes, p-values and false discovery rates.

Details

The new facilities for cleanly defining new stats and geoms added to 'ggplot2' in version 2.0.0 and the support for nested tibbles and new syntax for mapping computed values to aesthetics added to 'ggplot2' in version 3.0.0 are used in this package's code. This means that 'ggpmisc' (>= 0.3.0) requires version 3.0.0 or later of ggplot2 while 'ggpmisc' (< 0.3.0) requires version 2.0.0 or later of ggplot2.

Extensions provided:

- Function for conversion of time series data into tibbles that can be plotted with ggplot.
- ggplot() method for time series data.
- Stats for locating and tagging "peaks" and "valleys" (local or global maxima and minima).
- Stat for generating labels from a lm() model fit, including formatted equation. By default labels are expressions but tikz device is supported optionally with LaTeX formatted labels.
- Stats for extracting information from a any model fit supported by package 'broom'.
- Stats for filtering-out/filtering-in observations in regions of a panel or group where the density of observations is high.
- Geom for annotating plots with tables.

The stats for peaks and valleys are coded so as to work correctly both with numeric and POSIXct variables mapped to the x aesthetic. Special handling was needed as text labels are generated from the data.

Warning!

geom_null(), stat_debug_group(), stat_debug_panel(), geom_debug(), append_layers(), bottom_layer(), delete_layers(), extract_layers(), move_layers(), num_layers(), shift_layers(), top_layer() and which_layers() have been moved from package 'ggpmisc' into their own separate package 'gginnards-package.'
Acknowledgements

We thank Kamil Slowikowski not only for contributing ideas and code examples to this package but also for adding new features to his package 'ggrepel' that allow new use cases for `stat_dens2d_labels` from this package.

Note

The signatures of `stat_peaks()` and `stat_valleys()` are identical to those of `stat_peaks` and `stat_valleys` from package photobiology but the variables returned are a subset as values related to light spectra are missing. Furthermore the stats from package ggpmisc work correctly when the x aesthetic uses a date or datetime scale, while those from package photobiology do not generate correct labels in this case.

Author(s)

**Maintainer:** Pedro J. Aphalo <pedro.aphalo@helsinki.fi> (ORCID)

Other contributors:

- Kamil Slowikowski (ORCID) [contributor]
- Samer Mouksassi <samermouksassi@gmail.com> (ORCID) [contributor]

References

Package suite 'r4photobiology' web site at [https://www.r4photobiology.info/](https://www.r4photobiology.info/)
Package 'ggplot2' documentation at [https://ggplot2.tidyverse.org/](https://ggplot2.tidyverse.org/)
Package 'ggplot2' source code at [https://github.com/tidyverse/ggplot2](https://github.com/tidyverse/ggplot2)

See Also

Useful links:

- [https://docs.r4photobiology.info/ggpmisc/](https://docs.r4photobiology.info/ggpmisc/)
- [https://github.com/aphalo/ggpmisc](https://github.com/aphalo/ggpmisc)
- Report bugs at [https://github.com/aphalo/ggpmisc/issues](https://github.com/aphalo/ggpmisc/issues)

Examples

```r
library(tibble)

ggplot(lynx, as.numeric = FALSE) + geom_line() +
  stat_peaks(colour = "red") +
  stat_peaks(geom = "text", colour = "red", angle = 66,
             hjust = -0.1, x.label.fmt = "%Y") +
  ylim(NA, 8000)

formula <- y ~ poly(x, 2, raw = TRUE)

ggplot(cars, aes(speed, dist)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(aes(label = after_stat(eq.label)),
```
Moved

\begin{verbatim}
formula <- y ~ x
ggplot(PlantGrowth, aes(group, weight)) +
  stat_summary(fun.data = "mean_se") +
  stat_fit_tb(method = "lm",
    method.args = list(formula = formula),
    tb.type = "fit.anova",
    tb.vars = c(Term = "term", "df", "M.S." = "meansq",
      "italic(F)" = "statistic",
      "italic(p)" = "p.value"),
    tb.params = c("Group" = 1, "Error" = 2),
    table.theme = ttheme_gtbw(parse = TRUE)) +
  labs(x = "Group", y = "Dry weight of plants") +
  theme_classic()
\end{verbatim}

\section*{Moved to package ‘gginnards’}

\section*{Description}

Some stats, geoms and the plot layer manipulation functions have been moved from package ‘ggpmisc’ to a separate new package called ‘gginnards’.

\section*{Details}

To continue using any of these functions and methods, simply run at the R prompt or add to your script `library(gginnards)`, after installing package ‘gginnards’.

\section*{See Also}

`gginnards-package, geom_null, stat_debug_group, stat_debug_panel, geom_debug` and `delete_layers`.

\section*{outcome2factor}

\section*{Description}

Convert numeric ternary outcomes into a factor

\section*{Usage}

\begin{verbatim}
outcome2factor(x, n.levels = 3L)
threshold2factor(x, n.levels = 3L, threshold = 0)
\end{verbatim}
Arguments

- **x**: a numeric vector of -1, 0, and +1 values, indicating down-regulation, uncertain response or up-regulation, or a numeric vector that can be converted into such values using a pair of thresholds.
- **n.levels**: numeric Number of levels to create, either 3 or 2.
- **threshold**: numeric vector Range enclosing the values to be considered uncertain.

Details

These functions convert the numerically encoded values into a factor with the three levels "down", "uncertain" and "up", or into a factor with two levels de and uncertain as expected by default by scales `scale_colour_outcome`, `scale_fill_outcome` and `scale_shape_outcome`. When `n.levels = 2` both -1 and +1 are merged to the same level of the factor with label "de".

Note

These are convenience functions that only save some typing. The same result can be achieved by a direct call to `factor` and comparisons. These functions aim at making it easier to draw volcano and quadrant plots.

See Also

Other Functions for quadrant and volcano plots: `FC_format()`, `scale_colour_outcome()`, `scale_shape_outcome()`, `scale_y_Pvalue()`, `xy_outcomes2factor()`

Other scales for omics data: `scale_shape_outcome()`, `scale_x_logFC()`, `xy_outcomes2factor()`

Examples

```r
outcome2factor(c(-1, 1, 0, 1))
outcome2factor(c(-1, 1, 0, 1), n.levels = 2L)
threshold2factor(c(-0.1, -2, 0, +5))
threshold2factor(c(-0.1, -2, 0, +5), n.levels = 2L)
threshold2factor(c(-0.1, -2, 0, +5), threshold = c(-1, 1))
```

---

### quadrant_example.df

**Example gene expression data**

Description

A dataset containing reshaped and simplified output from an analysis of data from RNAseq done with package edgeR. Original data from gene expression in the plant species *Arabidopsis thaliana*.

Usage

`quadrant_example.df`
scale_colour_outcome

Format

A data.frame object with 6088 rows and 6 variables

See Also

Other Transcriptomics data examples: volcano_example.df

Examples

names(quadrant_example.df)
head(quadrant_example.df)

scale_colour_outcome  Colour and fill scales for ternary outcomes

Description

Manual scales for colour and fill aesthetics with defaults suitable for the three way outcome from some statistical tests.

Usage

scale_colour_outcome(
  ..., 
  name = "Outcome",
  ns.colour = "grey80",
  up.colour = "red",
  down.colour = "dodgerblue2",
  de.colour = "goldenrod",
  na.colour = "black",
  aesthetics = "colour"
)

scale_color_outcome(
  ..., 
  name = "Outcome",
  ns.colour = "grey80",
  up.colour = "red",
  down.colour = "dodgerblue2",
  de.colour = "goldenrod",
  na.colour = "black",
  aesthetics = "colour"
)

scale_fill_outcome(
  ..., 
  name = "Outcome",
  ns.colour = "grey80",
  up.colour = "red",
  down.colour = "dodgerblue2",
  de.colour = "goldenrod",
  na.colour = "black",
  aesthetics = "colour"
)
Arguments

... other named arguments passed to scale_manual.

name
The name of the scale, used for the axis-label.

ns.colour, down.colour, up.colour, de.colour
The colour definitions to use for each of the three possible outcomes.

na.colour
colour definition used for NA.

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").

Details

These scales only alter the breaks, values, and na.value default arguments of scale_colour_manual() and scale_fill_manual(). Please, see documentation for scale_manual for details.

See Also

Other Functions for quadrant and volcano plots: FC_format(), outcome2factor(), scale_shape_outcome(), scale_y_Pvalue(), xy_outcomes2factor()

Examples

set.seed(12346)
outcome <- sample(c(-1, 0, +1), 50, replace = TRUE)
my.df <- data.frame(x = rnorm(50),
                     y = rnorm(50),
                     outcome2 = outcome2factor(outcome, n.levels = 2),
                     outcome3 = outcome2factor(outcome))

ggplot(my.df, aes(x, y, colour = outcome3)) +
       geom_point() +
       scale_colour_outcome() +
       theme_bw()

ggplot(my.df, aes(x, y, colour = outcome2)) +
       geom_point() +
       scale_colour_outcome() +
       theme_bw()
```r
ggplot(my.df, aes(x, y, fill = outcome3)) +
  geom_point(shape = 21) +
  scale_fill_outcome() +
  theme_bw()
```

### scale_shape_outcome

**Shape scale for ternary outcomes**

**Description**

Manual scales for colour and fill aesthetics with defaults suitable for the three way outcome from some statistical tests.

**Usage**

```r
scale_shape_outcome(
  ..., 
  name = "Outcome",
  ns.shape = "circle filled",
  up.shape = "triangle filled",
  down.shape = "triangle down filled",
  de.shape = "square filled",
  na.shape = "cross"
)
```

**Arguments**

- `...` other named arguments passed to `scale_manual`.
- `name` The name of the scale, used for the axis-label.
- `ns.shape`, `down.shape`, `up.shape`, `de.shape` The shapes to use for each of the three possible outcomes.
- `na.shape` Shape used for NA.

**Details**

These scales only alter the values, and `na.value` default arguments of `scale_shape_manual()`.

Please, see documentation for `scale_manual` for details.

**See Also**

Other Functions for quadrant and volcano plots: `FC_format()`, `outcome2factor()`, `scale_colour_outcome()`, `scale_y_Pvalue()`, `xy_outcomes2factor()`

Other scales for omics data: `outcome2factor()`, `scale_x_logFC()`, `xy_outcomes2factor()`
Examples

```r
set.seed(12346)
outcome <- sample(c(-1, 0, +1), 50, replace = TRUE)
my.df <- data.frame(x = rnorm(50),
                    y = rnorm(50),
                    outcome2 = outcome2factor(outcome, n.levels = 2),
                    outcome3 = outcome2factor(outcome))

ggplot(my.df, aes(x, y, shape = outcome3)) +
  geom_point() +
  scale_shape_outcome() +
  theme_bw()

ggplot(my.df, aes(x, y, shape = outcome3)) +
  geom_point() +
  scale_shape_outcome(guide = FALSE) +
  theme_bw()

ggplot(my.df, aes(x, y, shape = outcome2)) +
  geom_point(size = 2) +
  scale_shape_outcome() +
  theme_bw()

ggplot(my.df, aes(x, y, shape = outcome3, fill = outcome2)) +
  geom_point() +
  scale_shape_outcome() +
  scale_fill_outcome() +
  theme_bw()

ggplot(my.df, aes(x, y, shape = outcome3, fill = outcome2)) +
  geom_point() +
  scale_shape_outcome(name = "direction") +
  scale_fill_outcome(name = "significance") +
  theme_bw()
```

---

**scale_x_logFC**

*Position scales for log fold change data*

**Description**

Continuous scales for x and y aesthetics with defaults suitable for values expressed as log2 fold change in data and fold-change in tick labels. Supports tick labels and data expressed in any combination of fold-change, log2 fold-change and log10 fold-change. Supports addition of units to axis labels passed as argument to the name formal parameter.
scale_x_logFC

Usage

scale_x_logFC(
  name = "Abundance of x%unit",
  breaks = NULL,
  labels = NULL,
  limits = symmetric_limits,
  oob = scales::squish,
  expand = expansion(mult = 0.15, add = 0),
  log.base.labels = FALSE,
  log.base.data = 2L,
  ...
)

scale_y_logFC(
  name = "Abundance of y%unit",
  breaks = NULL,
  labels = NULL,
  limits = symmetric_limits,
  oob = scales::squish,
  expand = expansion(mult = 0.15, add = 0),
  log.base.labels = FALSE,
  log.base.data = 2L,
  ...
)

Arguments

name The name of the scale without units, used for the axis-label.
breaks The positions of ticks or a function to generate them. Default varies depending on argument passed to log.base.labels. if supplied as a numeric vector they should be given using the data as passed to parameter data.
labels The tick labels or a function to generate them from the tick positions. The default is function that uses the arguments passed to log.base.data and log.base.labels to generate suitable labels.
limits One of: NULL to use the default scale range from ggplot2. A numeric vector of length two providing limits of the scale, using NA to refer to the existing minimum or maximum. A function that accepts the existing (automatic) limits and returns new limits. The default is function symmetric_limits() which keep 1 at the middle of the axis.
oob Function that handles limits outside of the scale limits (out of bounds). The default squishes out-of-bounds values to the boundary.
expand Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. The default is to expand the scale by 15% on each end for log-fold-data, so as to leave space for counts annotations.
log.base.labels, log.base.data

integer or logical Base of logarithms used to express fold-change values in tick labels and in data. Use FALSE for no logarithm transformation.

... other named arguments passed to scale_y_continuous.

Details

These scales only alter default arguments of scale_x_continuous() and scale_y_continuous(). Please, see documentation for scale_continuous for details. The name argument supports the use of "%unit" at the end of the string to automatically add a units string, otherwise user-supplied values for names, breaks, and labels work as usual. Tick labels are built based on the transformation already applied to the data (log2 by default) and a possibly different log transformation (default is fold-change with no transformation).

See Also

Other scales for omics data: outcome2factor(), scale_shape_outcome(), xy_outcomes2factor()

Examples

set.seed(12346)
my.df <- data.frame(x = rnorm(50, sd = 4), y = rnorm(50, sd = 4))

ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC() +
  scale_y_logFC()

ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC(labels = scales::trans_format(function(x) {log10(2^x)},
               scales::math_format())) +
  scale_y_logFC(labels = scales::trans_format(function(x) {log10(2^x)},
               scales::math_format()))

ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC(log.base.labels = 2) +
  scale_y_logFC(log.base.labels = 2)

ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC("A concentration%unit", log.base.labels = 10) +
  scale_y_logFC("B concentration%unit", log.base.labels = 10)

ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC("A concentration%unit", breaks = NULL) +
  scale_y_logFC("B concentration%unit", breaks = NULL)

# taking into account that data are expressed as log2 FC.
scale_y_Pvalue

Description

Scales for y aesthetic mapped to P-values as used in volcano plots with transcriptomics and metabolomics data.

Usage

scale_y_Pvalue(
  ..., 
  name = expression(italic(P) - plain(value)),
  trans = NULL,
  breaks = NULL,
  labels = NULL,
  limits = c(1, 1e-20),
  oob = NULL,
  expand = NULL
)
scale_y_Pvalue

scale_y_FDR(
  ..., 
  name = "False discovery rate", 
  trans = NULL, 
  breaks = NULL, 
  labels = NULL, 
  limits = c(1, 1e-10), 
  oob = NULL, 
  expand = NULL
)

scale_x_Pvalue(
  ..., 
  name = expression(italic(P) - plain(value)), 
  trans = NULL, 
  breaks = NULL, 
  labels = NULL, 
  limits = c(1, 1e-20), 
  oob = NULL, 
  expand = NULL
)

scale_x_FDR(
  ..., 
  name = "False discovery rate", 
  trans = NULL, 
  breaks = NULL, 
  labels = NULL, 
  limits = c(1, 1e-10), 
  oob = NULL, 
  expand = NULL
)

Arguments

... other named arguments passed to scale_y_continuous.
name The name of the scale without units, used for the axis-label.
trans Either the name of a transformation object, or the object itself. Use NULL for the default.
breaks The positions of ticks or a function to generate them. Default varies depending on argument passed to log_base.labels.
labels The tick labels or a function to generate them from the tick positions. The default is function that uses the arguments passed to log_base.data and log_base.labels to generate suitable labels.
limits Use one of: NULL to use the default scale range, a numeric vector of length two
Describing the function `stat_fit_augment`

- **Providing limits of the scale; NA to refer to the existing minimum or maximum; a function that accepts the existing (automatic) limits and returns new limits.**
- **oob** Function that handles limits outside of the scale limits (out of bounds). The default squishes out-of-bounds values to the boundary.
- **expand** Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. The default is to expand the scale by 15% on each end for log-fold-data, so as to leave space for counts annotations.

**Details**

These scales only alter default arguments of `scale_x_continuous()` and `scale_y_continuous()`. Please, see documentation for `scale_continuous` for details.

**See Also**

Other Functions for quadrant and volcano plots: `FC_format()`, `outcome2factor()`, `scale_colour_outcome()`, `scale_shape_outcome()`, `xy_outcomes2factor()`

**Examples**

```r
set.seed(12346)
my.df <- data.frame(x = rnorm(50, sd = 4),
                    y = 10^-runif(50, min = 0, max = 20))

ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC() +
  scale_y_Pvalue()

ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC() +
  scale_y_FDR(limits = c(NA, 1e-20))
```

**Description**

`stat_fit_augment` fits a model and returns a "tidy" version of the model’s data with prediction added, using ‘`augment()` methods from packages ‘`broom`, ‘`broom.mixed`, or other sources. The prediction can be added to the plot as a curve.
Usage

```r
stat_fit_augment(
  mapping = NULL,
  data = NULL,
  geom = "smooth",
  method = "lm",
  method.args = list(formula = y ~ x),
  augment.args = list(),
  level = 0.95,
  y.out = ".fitted",
  position = "identity",
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  ...
)
```

Arguments

- `mapping`: The aesthetic mapping, usually constructed with `aes` or `aes_`. Only needs to be set at the layer level if you are overriding the plot defaults.
- `data`: A layer specific dataset - only needed if you want to override the plot defaults.
- `geom`: The geometric object to use display the data
- `method`: character or function.
- `method.args`, `augment.args`: list of arguments to pass to `method` and to `broom:augment`.
- `level`: numeric Level of confidence interval to use (0.95 by default)
- `y.out`: character (or numeric) index to column to return as `y`.
- `position`: The position adjustment to use for overlapping points on this layer
- `na.rm`: logical indicating whether NA values should be stripped before the computation proceeds.
- `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.
- `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`.
- `...`: other arguments passed on to `layer`. This can include aesthetics whose values you want to set, not map. See `layer` for more details.

Details

`stat_fit_augment` together with `stat_fit_glance` and `stat_fit_tidy`, based on package 'broom' can be used with a broad range of model fitting functions as supported at any given time by 'broom'. In contrast to `stat_poly_eq` which can generate text or expression labels automatically, for these functions the mapping of aesthetic label needs to be explicitly supplied in the call, and labels built on the fly.
A ggplot statistic receives as data a data frame that is not the one passed as argument by the user, but instead a data frame with the variables mapped to aesthetics. In other words, it respects the grammar of graphics and consequently within arguments passed through method.args names of aesthetics like $x$ and $y$ should be used instead of the original variable names, while data is automatically passed the data frame. This helps ensure that the model is fitted to the same data as plotted in other layers.

Warning!

Not all 'glance()' methods are defined in package 'broom'. 'glance()' especializations for mixed models fits of classes 'lme', 'nlme', 'lme4', and many others are defined in package 'broom.mixed'.

Handling of grouping

stat_fit_augment applies the function given by method separately to each group of observations; in ggplot2 factors mapped to aesthetics generate a separate group for each level. Because of this, stat_fit_augment is not useful for annotating plots with results from t.test() or ANOVA or ANCOVA. In such cases use instead stat_fit_tb() which applies the model fitting per panel.

Computed variables

The output of augment() is returned as is, except for y which is set based on y.out and y.observed which preserves the y returned by the generics::augment methods. This renaming is needed so that the geom works as expected.

To explore the values returned by this statistic, which vary depending on the model fitting function and model formula we suggest the use of geom_debug. An example is shown below.

Note

The statistic stat_fit_augment can be used only with methods that accept formulas under any formal parameter name and a data argument. Use ggplot2::stat_smooth() instead of stat_fit_augment in production code if the additional features are not needed.

Although arguments passed to parameter augment.args will be passed to generics::augment() whether they are silently ignored or obeyed depends on each specialization of generics::augment(), so do carefully read the documentation for the version of generics::augment() corresponding to the 'method' used to fit the model.

See Also

broom and broom.mixed for details on how the tidying of the result of model fits is done.

Other ggplot statistics for model fits: stat_fit_deviations(), stat_fit_glance(), stat_fit_residuals(), stat_fit_tb(), stat_fit_tidy()

Examples

# package 'broom' needs to be installed to run these examples

if (requireNamespace("broom", quietly = TRUE)) {
  library(broom)
library(quantreg)

# Inspecting the returned data using geom_debug()
if (requireNamespace("gginnards", quietly = TRUE)) {
  library(gginnards)

  # Regression by panel
  ggplot(mtcars, aes(x = disp, y = mpg)) +
  geom_point(aes(colour = factor(cyl))) +
  stat_fit_augment(method = "lm",
                   method.args = list(formula = y ~ x),
                   geom = "debug",
                   summary.fun = colnames)
}

# Regression by panel example
ggplot(mtcars, aes(x = disp, y = mpg)) +
  geom_point(aes(colour = factor(cyl))) +
  stat_fit_augment(method = "lm",
                   method.args = list(formula = y ~ x))

# Residuals from regression by panel example
ggplot(mtcars, aes(x = disp, y = mpg)) +
  geom_hline(yintercept = 0, linetype = "dotted") +
  stat_fit_augment(geom = "point",
                   method = "lm",
                   method.args = list(formula = y ~ x),
                   y.out = ".resid")

# Regression by group example
ggplot(mtcars, aes(x = disp, y = mpg, colour = factor(cyl))) +
  geom_point() +
  stat_fit_augment(method = "lm",
                   method.args = list(formula = y ~ x))

# Residuals from regression by group example
ggplot(mtcars, aes(x = disp, y = mpg, colour = factor(cyl))) +
  geom_hline(yintercept = 0, linetype = "dotted") +
  stat_fit_augment(geom = "point",
                   method = "lm",
                   method.args = list(formula = y ~ x),
                   y.out = ".resid")

# Weighted regression example
ggplot(mtcars, aes(x = disp, y = mpg, weight = cyl)) +
  geom_point(aes(colour = factor(cyl))) +
  stat_fit_augment(method = "lm",
                   method.args = list(formula = y ~ x,
                   weights = quote(weight)))

# Residuals from weighted regression example
ggplot(mtcars, aes(x = disp, y = mpg, weight = cyl)) +
  geom_hline(yintercept = 0, linetype = "dotted") +
  stat_fit_augment(geom = "point",
                   method = "lm",
                   method.args = list(formula = y ~ x,
                   weights = quote(weight)))
method.args = list(formula = y ~ x,
                  weights = quote(weight)),
    y.out = ".resid")

# Quantile regression
ggplot(mtcars, aes(x = disp, y = mpg)) +
  geom_point() +
  stat_fit_augment(method = "rq",
                   label.y = "bottom")
}

stat_fit_deviations

Residuals from model fit as segments

Description

stat_fit_deviations fits a linear model and returns fitted values and residuals ready to be plotted as segments.

Usage

stat_fit_deviations(
  mapping = NULL,
  data = NULL,
  geom = "segment",
  method = "lm",
  method.args = list(),
  formula = NULL,
  position = "identity",
  na.rm = FALSE,
  orientation = NA,
  show.legend = FALSE,
  inherit.aes = TRUE,
  ...
)

Arguments

mapping The aesthetic mapping, usually constructed with aes or aes_. Only needs to be set at the layer level if you are overriding the plot defaults.
data A layer specific dataset - only needed if you want to override the plot defaults.
geom The geometric object to use display the data
method function or character If character, "lm", "rlm", "lqs" and "rq" are implemented. If a function, it must support parameters formula and data.
method.args named list with additional arguments.
formul a "formula" object. Using aesthetic names instead of original variable names.
position The position adjustment to use for overlapping points on this layer
na.rm a logical indicating whether NA values should be stripped before the computa-
tion proceeds.
orientation character Either "x" or "y" controlling the default for formula.
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.
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 should
not inherit behaviour from the default plot specification, e.g. borders.
...
other arguments passed on to layer. This can include aesthetics whose values
you want to set, not map. See layer for more details.

Details
This stat can be used to automatically highlight residuals as segments in a plot of a fitted model
equation. This stat only generates the residuals, the predicted values need to be separately added
to the plot, so to make sure that the same model formula is used in all steps it is best to save the
formula as an object and supply this object as argument to the different statistics.
A ggplot statistic receives as data a data frame that is not the one passed as argument by the user,
but instead a data frame with the variables mapped to aesthetics. In other words, it respects the
grammar of graphics and consequently within the model formula names of aesthetics like $x$ and
$y$ should be used instead of the original variable names. This helps ensure that the model is fitted
to the same data as plotted in other layers.

Computed variables
Data frame with same nrow as data as subset for each group containing five numeric variables.

- **x** x coordinates of observations
- **y.fitted** x coordinates of fitted values
- **y** y coordinates of observations
- **y.fitted** y coordinates of fitted values

To explore the values returned by this statistic we suggest the use of geom_debug. An example is
shown below, where one can also see in addition to the computed values the default mapping of the
fitted values to aesthetics xend and yend.

Note
In the case of method = "rq" quantiles are fixed at tau = 0.5 unless method.args has length > 0.
Parameter orientation is redundant as it only affects the default for formula but is included for
consistency with ggplot2.

See Also
Other ggplot statistics for model fits: stat_fit_augment(), stat_fit_glance(), stat_fit_residuals(),
stat_fit_tb(), stat_fit_tidy()
Examples

```r
# generate artificial data
library(MASS)
set.seed(4321)
x <- 1:100
y <- (x + x^2 + x^3) + rnorm(length(x), mean = 0, sd = mean(x^3) / 4)
my.data <- data.frame(x, y)

# plot residuals from linear model
ggplot(my.data, aes(x, y)) +
  geom_smooth(method = "lm", formula = y ~ x) +
  stat_fit_deviations(method = "lm", formula = y ~ x, colour = "red") +
  geom_point()

# plot residuals from linear model with y as explanatory variable
ggplot(my.data, aes(x, y)) +
  geom_smooth(method = "lm", formula = y ~ x, orientation = "y") +
  stat_fit_deviations(method = "lm", formula = x ~ y, colour = "red") +
  geom_point()

# as above using orientation
ggplot(my.data, aes(x, y)) +
  geom_smooth(method = "lm", orientation = "y") +
  stat_fit_deviations(orientation = "y", colour = "red") +
  geom_point()

# both regressions and their deviations
ggplot(my.data, aes(x, y)) +
  geom_smooth(method = "lm") +
  stat_fit_deviations(colour = "blue") +
  geom_smooth(method = "lm", orientation = "y", colour = "red") +
  stat_fit_deviations(orientation = "y", colour = "red") +
  geom_point()

# give a name to a formula
my.formula <- y ~ poly(x, 3, raw = TRUE)

# plot linear regression
ggplot(my.data, aes(x, y)) +
  geom_smooth(method = "lm", formula = my.formula) +
  stat_fit_deviations(formula = my.formula, colour = "red") +
  geom_point()

ggplot(my.data, aes(x, y)) +
  geom_smooth(method = "lm", formula = my.formula) +
  stat_fit_deviations(formula = my.formula, method = stats::lm, colour = "red") +
  geom_point()

# plot robust regression
ggplot(my.data, aes(x, y)) +
  stat_smooth(method = "rlm", formula = my.formula) +
  geom_point()
```
stat_fit_deviations(formula = my.formula, method = "rlm", colour = "red") + geom_point()

# plot robust regression with weights indicated by colour
my.data.outlier <- my.data
my.data.outlier[6, "y"] <- my.data.outlier[6, "y"] * 10
ggplot(my.data.outlier, aes(x, y)) +
  stat_smooth(method = MASS::rlm, formula = my.formula) +
  stat_fit_deviations(formula = my.formula, method = "rlm",
                      mapping = aes(colour = after_stat(weights)),
                      show.legend = TRUE) +
  scale_color_gradient(low = "red", high = "blue", limits = c(0, 1),
                      guide = "colourbar") +
  geom_point()

# plot quantile regression (= median regression)
ggplot(my.data, aes(x, y)) +
  stat_quantile(formula = my.formula, quantiles = 0.5) +
  stat_fit_deviations(formula = my.formula, method = "rq", colour = "red") +
  geom_point()

# plot quantile regression (= "quartile" regression)
ggplot(my.data, aes(x, y)) +
  stat_quantile(formula = my.formula, quantiles = 0.75) +
  stat_fit_deviations(formula = my.formula, colour = "red",
                      method = "rq", method.args = list(tau = 0.75)) +
  geom_point()

# inspecting the returned data
if (requireNamespace("gginnards", quietly = TRUE)) {
  library(gginnards)

  # plot, using geom_debug() to explore the after_stat data
  ggplot(my.data, aes(x, y)) +
  geom_smooth(method = "lm", formula = my.formula) +
  stat_fit_deviations(formula = my.formula, geom = "debug") +
  geom_point()

  ggplot(my.data.outlier, aes(x, y)) +
  stat_smooth(method = MASS::rlm, formula = my.formula) +
  stat_fit_deviations(formula = my.formula, method = "rlm", geom = "debug") +
  geom_point()
}

---

stat_fit_glance

One row summary data frame for a fitted model
Description

`stat_fit_glance` fits a model and returns a "tidy" version of the model's fit, using `glance()` methods from packages 'broom', 'broom.mixed', or other sources.

Usage

```r
stat_fit_glance(
  mapping = NULL,
  data = NULL,
  geom = "text_npc",
  method = "lm",
  method.args = list(formula = y ~ x),
  glance.args = list(),
  label.x = "left",
  label.y = "top",
  hstep = 0,
  vstep = 0.075,
  position = "identity",
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  ...
)
```

Arguments

- `mapping` The aesthetic mapping, usually constructed with `aes` or `aes_`. Only needs to be set at the layer level if you are overriding the plot defaults.
- `data` A layer specific data set - only needed if you want to override the plot defaults.
- `geom` The geometric object to use display the data
- `method` character or function.
- `method.args`, `glance.args` list of arguments to pass to `method` and to `[generics::glance()]`, respectively.
- `label.x, label.y` numeric with range 0..1 "normalized parent coordinates" (npc units) or character if using `geom_text_npc()` or `geom_label_npc()`. If using `geom_text()` or `geom_label()` numeric in native data units. If too short they will be recycled.
- `hstep, vstep` numeric in npc units, the horizontal and vertical step used between labels for different groups.
- `position` The position adjustment to use for overlapping points on this layer
- `na.rm` a logical indicating whether NA values should be stripped before the computation proceeds.
- `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.
- `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`.
... other arguments passed on to `layer`. This can include aesthetics whose values you want to set, not map. See `layer` for more details.

Details

`stat_fit_glance` together with `stat_fit_tidy` and `stat_fit_augment`, based on package 'broom' can be used with a broad range of model fitting functions as supported at any given time by package 'broom'. In contrast to `stat_poly_eq` which can generate text or expression labels automatically, for these functions the mapping of aesthetic `label` needs to be explicitly supplied in the call, and labels built on the fly.

A ggplot statistic receives as data a data frame that is not the one passed as argument by the user, but instead a data frame with the variables mapped to aesthetics. In other words, it respects the grammar of graphics and consequently within arguments passed through `method.args` names of aesthetics like $x$ and $y$ should be used instead of the original variable names, while data is automatically passed the data frame. This helps ensure that the model is fitted to the same data as plotted in other layers.

Value

The output of the `glance()` methods is returned almost as is in the data object, as a data frame. The names of the columns in the returned data are consistent with those returned by method `glance()` from package 'broom', that will frequently differ from the name of values returned by the print methods corresponding to the fit or test function used. To explore the values returned by this statistic including the name of variables/columns, which vary depending on the model fitting function and model formula we suggest the use of `geom_debug`. An example is shown below.

Warning!

Not all `glance()` methods are defined in package 'broom'. `glance()` especializations for mixed models fits of classes `lme`, `nlme`, `lme4`, and many others are defined in package 'broom.mixed'.

Handling of grouping

`stat_fit_glance` applies the function given by `method` separately to each group of observations, and factors mapped to aesthetics generate a separate group for each factor level. Because of this, `stat_fit_glance` is not useful for annotating plots with results from `t.test()`, ANOVA or ANCOVA. In such cases use the `stat_fit_tb()` statistic which applies the model fitting per panel.

Model formula required

The current implementation works only with methods that accept a formula as argument and which have a data parameter through which a data frame can be passed. For example, `lm()` should be used with the formula interface, as the evaluation of $x$ and $y$ needs to be delayed until the internal object of the ggplot is available. With some methods like `cor.test()` the data embedded in the "ggplot" object cannot be automatically passed as argument for the data parameter of the test or model fit function.
Note

Although arguments passed to parameter glance.args will be passed to generics::glance() whether they are silently ignored or obeyed depends on each specialization of glance(), so do carefully read the documentation for the version of glance() corresponding to the ‘method’ used to fit the model.

See Also

broom and broom.mixed for details on how the tidying of the result of model fits is done.

Other ggplot statistics for model fits: stat_fit_augment(), stat_fit_deviations(), stat_fit_residuals(), stat_fit_tb(), stat_fit_tidy()

Examples

# package 'broom' needs to be installed to run these examples

if (requireNamespace("broom", quietly = TRUE)) {
  library(broom)
  library(quantreg)

  # Inspecting the returned data using geom_debug()
  if (requireNamespace("gginnards", quietly = TRUE)) {
    library(gginnards)

    ggplot(mtcars, aes(x = disp, y = mpg)) +
      stat_smooth(method = "lm") +
      geom_point(aes(colour = factor(cyl))) +
      stat_fit_glance(method = "lm",
                       method.args = list(formula = y ~ x),
                       geom = "debug")
  }

  # Regression by panel example
  ggplot(mtcars, aes(x = disp, y = mpg)) +
    stat_smooth(method = "lm") +
    geom_point(aes(colour = factor(cyl))) +
    stat_fit_glance(method = "lm",
                    label.y = "bottom",
                    method.args = list(formula = y ~ x),
                    mapping = aes(label = sprintf("r^2="~%.3f~italic(P)="~%.2g",
                                                  after_stat(r.squared), after_stat(p.value))),
                    parse = TRUE)

  # Regression by group example
  ggplot(mtcars, aes(x = disp, y = mpg, colour = factor(cyl))) +
    stat_smooth(method = "lm") +
    geom_point() +
    stat_fit_glance(method = "lm",
                    label.y = "bottom",
                    method.args = list(formula = y ~ x),
                    mapping = aes(label = sprintf("r^2="~%.3f~italic(P)="~%.2g",
                                                  after_stat(r.squared), after_stat(p.value))),
                    parse = TRUE)
}
parse = TRUE)

# Weighted regression example
ggplot(mtcars, aes(x = disp, y = mpg, weight = cyl)) +
  stat_smooth(method = "lm") +
  geom_point(aes(colour = factor(cyl))) +
  stat_fit_glance(method = "lm",
                  label.y = "bottom",
                  method.args = list(formula = y ~ x, weights = quote(weight)),
                  mapping = aes(label = sprintf("\text{\texttt{Var}}^2 = %.3f; \text{\textit{P}} = %.2g",
                                               after_stat(r.squared), after_stat(p.value))),
                  parse = TRUE)

# correlation test
ggplot(mtcars, aes(x = disp, y = mpg)) +
  geom_point() +
  stat_fit_glance(method = "cor.test",
                  label.y = "bottom",
                  method.args = list(formula = ~ x + y),
                  mapping = aes(label = sprintf("r[Pearson] = %.3f; \text{\textit{P}} = %.2g",
                                               after_stat(estimate), after_stat(p.value))),
                  parse = TRUE)

ggplot(mtcars, aes(x = disp, y = mpg)) +
  geom_point() +
  stat_fit_glance(method = "cor.test",
                  label.y = "bottom",
                  method.args = list(formula = ~ x + y, method = "spearman", exact = FALSE),
                  mapping = aes(label = sprintf("r[Spearman] = %.3f; \text{\textit{P}} = %.2g",
                                               after_stat(estimate), after_stat(p.value))),
                  parse = TRUE)

# Quantile regression by group example
ggplot(mtcars, aes(x = disp, y = mpg)) +
  stat_smooth(method = "lm") +
  geom_point() +
  stat_fit_glance(method = "rq",
                  label.y = "bottom",
                  method.args = list(formula = y ~ x),
                  mapping = aes(label = sprintf("AIC = %.3g, BIC = %.3g",
                                               after_stat(AIC), after_stat(BIC))))

stat_fit_residuals  
Residuals from a model fit

Description

stat_fit_residuals fits a linear model and returns residuals ready to be plotted as points.
Usage

stat_fit_residuals(
  mapping = NULL,
  data = NULL,
  geom = "point",
  method = "lm",
  method.args = list(),
  formula = NULL,
  resid.type = NULL,
  position = "identity",
  na.rm = FALSE,
  orientation = NA,
  show.legend = FALSE,
  inherit.aes = TRUE,
...
)

Arguments

mapping  The aesthetic mapping, usually constructed with \texttt{aes} or \texttt{aes_}. Only needs to be set at the layer level if you are overriding the plot defaults.
data  A layer specific dataset - only needed if you want to override the plot defaults.
geom  The geometric object to use display the data
method  function or character If character, "lm", "rlm", and "rq" are implemented. If a function, it must support parameters \texttt{formula} and \texttt{data}.
method.args  named list with additional arguments.
formula  a "formula" object. Using aesthetic names instead of original variable names.
resid.type  character passed to \texttt{residuals()} as argument for \texttt{type}.
position  The position adjustment to use for overlapping points on this layer
na.rm  a logical indicating whether NA values should be stripped before the computation proceeds.
orientation  character Either "x" or "y" controlling the default for \texttt{formula}.
show.legend  logical. Should this layer be included in the legends? \texttt{NA}, the default, includes if any aesthetics are mapped. \texttt{FALSE} never includes, and \texttt{TRUE} always includes.
inherit.aes  If \texttt{FALSE}, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and should not inherit behaviour from the default plot specification, e.g. \texttt{borders}.
...
other arguments passed on to \texttt{layer}. This can include aesthetics whose values you want to set, not map. See \texttt{layer} for more details.

Details

This stat can be used to automatically plot residuals as points in a plot. At the moment it supports only linear models fitted with function \texttt{lm()}. This stat only generates the residuals.
A ggplot statistic receives as data a data frame that is not the one passed as argument by the user, but instead a data frame with the variables mapped to aesthetics. In other words, it respects the grammar of graphics and consequently within the model formula names of aesthetics like $x$ and $y$ should be used instead of the original variable names, while data is automatically passed the data frame. This helps ensure that the model is fitted to the same data as plotted in other layers.

**Computed variables**

Data frame with same nrow as data as subset for each group containing five numeric variables.

- **x**: x coordinates of observations
- **y.resid**: residuals from fitted values
- **y.resid.abs**: absolute residuals from the fit

By default `stat(y.resid)` is mapped to the y aesthetic.

**Note**

Parameter orientation is redundant as it only affects the default for formula but is included for consistency with `ggplot2`.

**See Also**

Other ggplot statistics for model fits: `stat_fit_augment()`, `stat_fit_deviations()`, `stat_fit_glance()`, `stat_fit_tb()`, `stat_fit_tidy()`

**Examples**

```r
# generate artificial data
set.seed(4321)
x <- 1:100
y <- (x + x^2 + x^3) + rnorm(length(x), mean = 0, sd = mean(x^3) / 4)
my.data <- data.frame(x, y)

# plot residuals from linear model
ggplot(my.data, aes(x, y)) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  stat_fit_residuals(formula = y ~ x)

# plot residuals from linear model with y as explanatory variable
ggplot(my.data, aes(x, y)) +
  geom_vline(xintercept = 0, linetype = "dashed") +
  stat_fit_residuals(formula = x ~ y) +
  coord_flip()

# give a name to a formula
my.formula <- y ~ poly(x, 3, raw = TRUE)

# plot residuals from linear model
ggplot(my.data, aes(x, y)) +
```

```r
```
stat_fit_residuals

geom_hline(yintercept = 0, linetype = "dashed") +
stat_fit_residuals(formula = my.formula) +
coord_flip()

ggplot(my.data, aes(x, y)) +
geom_hline(yintercept = 0, linetype = "dashed") +
stat_fit_residuals(formula = my.formula, resid.type = "response")

# plot residuals from robust regression
ggplot(my.data, aes(x, y)) +
geom_hline(yintercept = 0, linetype = "dashed") +
stat_fit_residuals(formula = my.formula, method = "rlm")

# plot residuals with weights indicated by colour
my.data.outlier <- my.data
my.data.outlier[6, "y"] <- my.data.outlier[6, "y"] * 10
ggplot(my.data.outlier, aes(x, y)) +
stat_fit_residuals(formula = my.formula, method = "rlm",
mapping = aes(colour = after_stat(weights)),
show.legend = TRUE) +
scale_color_gradient(low = "red", high = "blue", limits = c(0, 1),
guide = "colourbar")

# plot weighted residuals with weights indicated by colour
ggplot(my.data.outlier) +
stat_fit_residuals(formula = my.formula, method = "rlm",
mapping = aes(x = x,
y = stage(start = y, after_stat = y * weights),
colour = after_stat(weights)),
show.legend = TRUE) +
scale_color_gradient(low = "red", high = "blue", limits = c(0, 1),
guide = "colourbar")

# plot residuals from quantile regression (median)
ggplot(my.data, aes(x, y)) +
geom_hline(yintercept = 0, linetype = "dashed") +
stat_fit_residuals(formula = my.formula, method = "rq")

# plot residuals from quantile regression (upper quartile)
ggplot(my.data, aes(x, y)) +
geom_hline(yintercept = 0, linetype = "dashed") +
stat_fit_residuals(formula = my.formula, method = "rq",
method.args = list(tau = 0.75))

# inspecting the returned data
if (requireNamespace("gginnards", quietly = TRUE)) {
  library(gginnards)
  ggplot(my.data, aes(x, y)) +
  stat_fit_residuals(formula = my.formula, resid.type = "working",
                    geom = "debug")
}

}
stat_fit_residuals(formula = my.formula, method = "rlm", 
geom = "debug")

stat_fit_tb

Model-fit summary or ANOVA

Description

stat_fit_tb fits a model and returns a "tidy" version of the model’s summary or ANOVA table, using ‘tidy()’ methods from packages ‘broom’, ‘broom.mixed’, or other sources. The annotation is added to the plots in tabular form.

Usage

stat_fit_tb(
  mapping = NULL, 
  data = NULL, 
  geom = "table_npc", 
  method = "lm", 
  method.args = list(formula = y ~ x), 
  tidy.args = list(), 
  tb.type = "fit.summary", 
  tb.vars = NULL, 
  tb.params = NULL, 
  digits = 3, 
  p.digits = digits, 
  label.x = "center", 
  label.y = "top", 
  label.x.npc = NULL, 
  label.y.npc = NULL, 
  position = "identity", 
  table.theme = NULL, 
  table.rownames = FALSE, 
  table.colnames = TRUE, 
  table.hjust = 1, 
  parse = FALSE, 
  na.rm = FALSE, 
  show.legend = FALSE, 
  inherit.aes = TRUE, 
  ... 
)

Arguments

mapping The aesthetic mapping, usually constructed with aes or aes_. Only needs to be set at the layer level if you are overriding the plot defaults.
stat_fit_tb

**data**  A layer specific dataset, only needed if you want to override the plot defaults.

**geom**  The geometric object to use to display the data.

**method**  character.

**method.args, tidy.args**  lists of arguments to pass to method and to tidy().

**tb.type**  character One of "fit.summary", "fit.anova" or "fit.coefs".

**tb.vars, tb.params**  character or numeric vectors, optionally named, used to select and/or rename the columns or the parameters in the table returned.

**digits**  integer indicating the number of significant digits to be used for all numeric values in the table.

**p.digits**  integer indicating the number of decimal places to round p-values to, with those rounded to zero displayed as the next larger possible value preceded by "<". If p.digits is outside the range 1..22 no rounding takes place.

**label.x, label.y**  numeric Coordinates (in data units) to be used for absolute positioning of the output. If too short they will be recycled.

**label.x.npc, label.y.npc**  numeric with range 0..1 or character. Coordinates to be used for positioning the output, expressed in "normalized parent coordinates" or character string. If too short they will be recycled.

**position**  The position adjustment to use for overlapping points on this layer.

**table.theme**  NULL, list or function A gridExtra theme definition, or a constructor for a theme or NULL for default.

**table.rownames, table.colnames**  logical flag to enable or disabling printing of row names and column names.

**table.hjust**  numeric Horizontal justification for the core and column headings of the table.

**parse**  If TRUE, the labels will be parsed into expressions and displayed as described in ?plotmath.

**na.rm**  a logical indicating whether NA values should be stripped before the computation proceeds.

**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.

**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.

**...**  other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.

**Details**

**stat_fit_tb** Applies a model fitting function per panel, using the grouping factors from aesthetic mappings in the fitted model. This is suitable, for example for analysis of variance used to test for differences among groups.
The argument to method can be any fit method for which a suitable tidy() method is available, including non-linear regression. Fit methods retain their default arguments unless overridden.

A ggplot statistic receives as data a data frame that is not the one passed as argument by the user, but instead a data frame with the variables mapped to aesthetics. In other words, it respects the grammar of graphics and consequently within arguments passed through method.args names of aesthetics like $x$ and $y$ should be used instead of the original variable names, while data is automatically passed the data frame. This helps ensure that the model is fitted to the same data as plotted in other layers.

**Computed variables**

The output of tidy() is returned as a single "cell" in a tibble (i.e. a tibble nested within a tibble). The returned data object contains a single, containing the result from a single model fit to all data in a panel. If grouping is present, it is ignored.

To explore the values returned by this statistic, which vary depending on the model fitting function and model formula we suggest the use of geom_debug.

**See Also**

broom and broom.mixed for details on how the tidying of the result of model fits is done. See geom_table for details on how inset tables respond to mapped aesthetics and table themes. For details on predefined table themes see ttheme_gtdefault.

Other ggplot statistics for model fits: stat_fit_augment(), stat_fit_deviations(), stat_fit_glance(), stat_fit_residuals(), stat_fit_tidy()

**Examples**

```r
if (requireNamespace("broom", quietly = TRUE)) {
  library(broom)

  # data for examples
  x <- c(44.4, 45.9, 41.9, 53.3, 44.7, 44.1, 50.7, 45.2, 60.1)
  covariate <- sqrt(x) + rnorm(9)
  group <- factor(c(rep("A", 4), rep("B", 5)))
  my.df <- data.frame(x, group, covariate)

  # Linear regression fit summary, by default
  ggplot(my.df, aes(covariate, x)) +
    geom_point() +
    stat_fit_tb() +
    expand_limits(y = 70)

  # Linear regression fit summary, by default
  ggplot(my.df, aes(covariate, x)) +
    geom_point() +
    stat_fit_tb(digits = 2, p.digits = 4) +
    expand_limits(y = 70)
```
# Linear regression fit summary
```r
ggplot(my.df, aes(covariate, x)) +
  geom_point() +
  stat_fit_tb(tb.type = "fit.summary") +
  expand_limits(y = 70)
```

# Linear regression ANOVA table
```r
ggplot(my.df, aes(covariate, x)) +
  geom_point() +
  stat_fit_tb(tb.type = "fit.anova") +
  expand_limits(y = 70)
```

# Linear regression fit coefficients
```r
ggplot(my.df, aes(covariate, x)) +
  geom_point() +
  stat_fit_tb(tb.type = "fit.coefs") +
  expand_limits(y = 70)
```

# Polynomial regression
```r
ggplot(my.df, aes(covariate, x)) +
  geom_point() +
  stat_fit_tb(method.args = list(formula = y ~ poly(x, 2))) +
  expand_limits(y = 70)
```

# Polynomial regression with renamed parameters
```r
ggplot(my.df, aes(covariate, x)) +
  geom_point() +
  stat_fit_tb(method.args = list(formula = y ~ poly(x, 2)),
              tb.params = c("x^0" = 1, "x^1" = 2, "x^2" = 3),
              parse = TRUE) +
  expand_limits(y = 70)
```

# Polynomial regression with renamed parameters and columns
```r
ggplot(my.df, aes(covariate, x)) +
  geom_point() +
  stat_fit_tb(method.args = list(formula = y ~ poly(x, 2)),
              tb.params = c("x^0" = 1, "x^1" = 2, "x^2" = 3),
              tb.vars = c("Term" = 1, "Estimate" = 2, "S.E." = 3,
                         "italic(F)-value" = 4, "italic(P)-value" = 5),
              parse = TRUE) +
  expand_limits(y = 70)
```

# ANOVA summary
```r
ggplot(my.df, aes(group, x)) +
  geom_point() +
  stat_fit_tb(tb.type = "fit.anova") +
  expand_limits(y = 70)
```

# ANOVA table
```r
ggplot(my.df, aes(group, x)) +
  geom_point() +
  stat_fit_tb(tb.type = "fit.anova") +
  expand_limits(y = 70)
```
expand_limits(y = 70)

# ANOVA table with renamed and selected columns
# using column names
ggplot(my.df, aes(group, x)) +
  geom_point() +
  stat_fit_tb(tb.type = "fit.anova",
    tb.vars = c("Effect" = "term", "df", "italic(F)" = "statistic",
              "italic(P)" = "p.value"),
    parse = TRUE)

# ANOVA table with renamed and selected columns
# using column names with partial matching
ggplot(my.df, aes(group, x)) +
  geom_point() +
  stat_fit_tb(tb.type = "fit.anova",
    tb.vars = c("Effect" = "term", "df", "italic(F)" = "stat",
                "italic(P)" = "p"),
    parse = TRUE)

# ANOVA summary
ggplot(my.df, aes(group, x)) +
  geom_point() +
  stat_fit_tb() +
  expand_limits(y = 70)

# ANCOVA (covariate not plotted)
ggplot(my.df, aes(group, x, z = covariate)) +
  geom_point() +
  stat_fit_tb(method.args = list(formula = y ~ x + z),
              tb.vars = c("Effect" = "term", "italic(F)" = "statistic",
                         "italic(P)" = "p.value"),
              parse = TRUE)

# t-test
ggplot(my.df, aes(group, x)) +
  geom_point() +
  stat_fit_tb(method = "t.test",
              tb.vars = c("italic(t)" = "statistic", "italic(P)" = "p.value"),
              parse = TRUE)

# t-test (equal variances assumed)
ggplot(my.df, aes(group, x)) +
  geom_point() +
  stat_fit_tb(method = "t.test",
              method.args = list(formula = y ~ x, var.equal = TRUE),
              tb.vars = c("italic(t)" = "statistic", "italic(P)" = "p.value"),
              parse = TRUE)

# Linear regression using a table theme
ggplot(my.df, aes(covariate, x)) +
  geom_point() +
  stat_fit_tb(table.theme = ttheme_gtlight) +
  expand_limits(y = 70)
stat_fit_tidy

One row data frame with fitted parameter estimates

Description

stat_fit_tidy fits a model and returns a "tidy" version of the model's summary, using 'tidy()' methods from packages 'broom', 'broom.mixed', or other sources. To add the summary in tabular form use stat_fit_tb instead of this statistic. When using stat_fit_tidy() you will most likely want to change the default mapping for label.

Usage

stat_fit_tidy(
mapping = NULL,
data = NULL,
geom = "text_npc",
method = "lm",
method.args = list(formula = y ~ x),
tidy.args = list(),
label.x = "left",
label.y = "top",
hstep = 0,
vstep = NULL,
sanitize.names = FALSE,
position = "identity",
na.rm = FALSE,
show.legend = FALSE,
inherit.aes = TRUE,
...
)

Arguments

mapping The aesthetic mapping, usually constructed with aes or aes_. Only needs to be set at the layer level if you are overriding the plot defaults.
data A layer specific dataset - only needed if you want to override the plot defaults.
geom The geometric object to use display the data
method character or function.
method.args, tidy.args list of arguments to pass to method, and to generics::tidy, respectively.
stat_fit_tidy

**label.x, label.y**

numeric with range 0..1 or character. Coordinates to be used for positioning the output, expressed in "normalized parent coordinates" or character string. If too short they will be recycled.

**hstep, vstep**

numeric in npc units, the horizontal and vertical step used between labels for different groups.

**sanitize.names**

logical If true sanitize column names in the returned data with R’s make.names() function.

**position**

The position adjustment to use for overlapping points on this layer.

**na.rm**

a logical indicating whether NA values should be stripped before the computation proceeds.

**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.

**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.

**...**

other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.

**Details**

stat_fit_tidy together with stat_fit_glance and stat_fit_augment, based on package ‘broom’ can be used with a broad range of model fitting functions as supported at any given time by ‘broom’. In contrast to stat_poly_eq which can generate text or expression labels automatically, for these functions the mapping of aesthetic label needs to be explicitly supplied in the call, and labels built on the fly.

A ggplot statistic receives as data a data frame that is not the one passed as argument by the user, but instead a data frame with the variables mapped to aesthetics. In other words, it respects the grammar of graphics and consequently within arguments passed through method. args names of aesthetics like $x$ and $y$ should be used instead of the original variable names, while data is automatically passed the data frame. This helps ensure that the model is fitted to the same data as plotted in other layers.

**Value**

The output of tidy() is returned after reshaping it into a single row. Grouping is respected, and the model fitted separately to each group of data. The returned data object has one row for each group within a panel. To use the intercept, note that output of tidy() is renamed from (Intercept) to Intercept. Otherwise, the names of the columns in the returned data are based on those returned by the tidy() method for the model fit class returned by the fit function. These will frequently differ from the name of values returned by the print methods corresponding to the fit or test function used. To explore the values returned by this statistic including the name of variables/columns, which vary depending on the model fitting function and model formula, we suggest the use of geom_debug. An example is shown below. Names of columns as returned by default are not always syntactically valid R names making it necessary to use back ticks to access them. Syntactically valid names are guaranteed if sanitize.names = TRUE is added to the call.
To explore the values returned by this statistic, which vary depending on the model fitting function and model formula we suggest the use of `geom_debug`. An example is shown below.

**Warning!**

Not all `glance()` methods are defined in package `broom`. `glance()` specializations for mixed models fits of classes `lme`, `nlme`, `lme4`, and many others are defined in package `broom.mixed`.

**Handling of grouping**

`stat_fit_tidy` applies the function given by method separately to each group of observations; in ggplot2 factors mapped to aesthetics generate a separate group for each level. Because of this, `stat_fit_tidy` is not useful for annotating plots with results from `t.test()` or ANOVA or ANCOVA. In such cases use instead `stat_fit_tb()` which applies the model fitting per panel.

**Note**

The statistic `stat_fit_tidy` can be used only with methods that accept formulas under any formal parameter name and a `data` argument. Use `ggplot2::stat_smooth()` instead of `stat_fit_augment` in production code if the additional features are not needed.

Although arguments passed to parameter `tidy.args` will be passed to `[generics::tidy()]` whether they are silently ignored or obeyed depends on each specialization of `[tidy()]`, so do carefully read the documentation for the version of `[tidy()]` corresponding to the `method` used to fit the model. You will also need to manually install the package, such as `broom`, where the tidier you intend to use are defined.

**See Also**

`broom` and `broom.mixed` for details on how the tidying of the result of model fits is done.

Other ggplot statistics for model fits: `stat_fit_augment()`, `stat_fit_deviations()`, `stat_fit_glance()`, `stat_fit_residuals()`, `stat_fit_tb()`

**Examples**

```r
# package 'broom' needs to be installed to run these examples

if (requireNamespace("broom", quietly = TRUE)) {
  library(broom)
  library(quantreg)

  # Inspecting the returned data using geom_debug()
  if (requireNamespace("gginnards", quietly = TRUE)) {
    library(gginnards)

    # This provides a quick way of finding out the names of the variables that
    # are available for mapping to aesthetics. This is specially important for
    # this stat as these names depend on the specific tidy() method used, which
    # depends on the method used, such as lm(), used to fit the model.

    # Regression by panel, default column names
```
```r
# Regression by panel example
ggplot(mtcars, aes(x = disp, y = mpg, colour = factor(cyl))) +
  stat_smooth(method = "lm") +
  geom_point() +
  stat_fit_tidy(method = "lm",
               label.x = "right",
               method.args = list(formula = y ~ x - I(x^2)),
               mapping = aes(label = sprintf("Slope = %.3g, p-value = %.3g",
                                             after_stat(x_estimate),
                                             after_stat(x_p.value))))

# Weighted regression example
ggplot(mtcars, aes(x = disp, y = mpg, weight = cyl)) +
  stat_smooth(method = "lm") +
  geom_point(aes(colour = factor(cyl))) +
  stat_fit_tidy(method = "lm",
                label.x = "right",
                method.args = list(formula = y ~ x, weights = quote(weight)),
                mapping = aes(label = sprintf("Slope = %.3g\n p-value = %.3g",
                                              after_stat(x_estimate),
                                              after_stat(x_p.value))))

# Correlation test
ggplot(mtcars, aes(x = disp, y = mpg)) +
  stat_smooth(method = "lm") +
  geom_point()
```
stat_peaks

Local maxima (peaks) or minima (valleys)

Description

stat_peaks finds at which x positions local y maxima are located and stat_valleys finds at which x positions local y minima are located. Both stats return a subset of data with rows matching for peaks or valleys with formatted character labels added. The formatting is determined by a format string compatible with `sprintf()` or `strftime()`.

Usage

```r
stat_peaks(
  mapping = NULL,
  data = NULL,
  geom = "point",
  span = 5,
  ignore_threshold = 0,
  strict = FALSE,
  label.fmt = NULL,
  x.label.fmt = NULL,
  y.label.fmt = NULL,
  position = "identity",
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  ...
)```
stat_peaks()

stat_valleys(
    mapping = NULL,
    data = NULL,
    geom = "point",
    span = 5,
    ignore_threshold = 0,
    strict = FALSE,
    label.fmt = NULL,
    x.label.fmt = NULL,
    y.label.fmt = NULL,
    position = "identity",
    na.rm = FALSE,
    show.legend = FALSE,
    inherit.aes = TRUE,
    ...
)

Arguments

mapping  The aesthetic mapping, usually constructed with aes or aes_. Only needs to be set at the layer level if you are overriding the plot defaults.
data      A layer specific dataset - only needed if you want to override the plot defaults.
geom      The geometric object to use display the data.
span      a peak is defined as an element in a sequence which is greater than all other elements within a window of width span centered at that element. The default value is 5, meaning that a peak is bigger than two consecutive neighbors on each side. A NULL value for span is taken as a span covering the whole of the data range.
ignore_threshold numeric value between 0.0 and 1.0 indicating the size threshold below which peaks will be ignored.
strict    logical flag: if TRUE, an element must be strictly greater than all other values in its window to be considered a peak. Default: FALSE.
label.fmt character string giving a format definition for converting values into character strings by means of function sprintf or strftime, its use is deprecated.
x.label.fmt character string giving a format definition for converting $x$-values into character strings by means of function sprintf or strftime. The default argument varies depending on the scale in use.
y.label.fmt character string giving a format definition for converting $y$-values into character strings by means of function sprintf.
position  The position adjustment to use for overlapping points on this layer.
na.rm     a logical value indicating whether NA values should be stripped before the computation proceeds.
stat_peaks

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.

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.

... other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.

Details

These stats use geom_point by default as it is the geom most likely to work well in almost any situation without need of tweaking. The default aesthetics set by these stats allow their direct use with geom_text, geom_label, geom_line, geom_rug, geom_hline and geom_vline. The formatting of the labels returned can be controlled by the user.

The default for parameter strict is TRUE in functions splus2R::peaks() and find_peaks(), while the default is FALSE in stat_peaks() and in stat_valleys().

Returned and computed variables

x x-value at the peak (or valley) as numeric
y y-value at the peak (or valley) as numeric
x.label x-value at the peak (or valley) as character
y.label y-value at the peak (or valley) as character

Note

These stats check the scale of the x aesthetic and if it is Date or Datetime they correctly generate the labels by transforming the numeric x values to Date or POSIXct objects, respectively. In which case the x.label.fmt must follow the syntax supported by strftime() rather than by sprintf(). These stats work nicely together with geoms geom_text_repel and geom_label_repel from package ggrepel to solve the problem of overlapping labels by displacing them. Alternatively, to discard overlapping labels use check_overlap = TRUE as argument to geom_text. By default the labels are character values suitable to be plotted as is, but with a suitable format passed as argument to label.fmt labels suitable for parsing by the geoms (e.g. into expressions containing Greek letters, super- or subscripts, maths symbols or maths constructs) can be also easily obtained.

Examples

# lynx is a time.series object
lynx_num.df <-
  try_tibble(lynx,
     col.names = c("year", "lynx"),
     as.numeric = TRUE) # years -> as numeric

ggplot(lynx_num.df, aes(year, lynx)) +
  geom_line() +
  stat_peaks(colour = "red") +
  stat_valleys(colour = "blue")
ggplot(lynx_num.df, aes(year, lynx)) +
  geom_line() +
  stat_peaks(colour = "red") +
  stat_peaks(colour = "red", geom = "rug")

ggplot(lynx_num.df, aes(year, lynx)) +
  geom_line() +
  stat_peaks(colour = "red") +
  stat_peaks(colour = "red", geom = "rug")

lynx_datetime.df <-
  try_tibble(lynx,
    col.names = c("year", "lynx")) # years -> POSIXct

ggplot(lynx_datetime.df, aes(year, lynx)) +
  geom_line() +
  stat_peaks(colour = "red") +
  stat_valleys(colour = "blue")

ggplot(lynx_datetime.df, aes(year, lynx)) +
  geom_line() +
  stat_peaks(colour = "red") +
  stat_peaks(colour = "red",
    geom = "text",
    hjust = -0.1,
    x.label.fmt = "%Y",
    angle = 33)

ggplot(lynx_datetime.df, aes(year, lynx)) +
  geom_line() +
  stat_peaks(colour = "red") +
  stat_peaks(colour = "red",
    geom = "text_linked",
    position = position_nudge_keep(x = 0, y = 200),
    hjust = -0.1,
    x.label.fmt = "%Y",
    angle = 90) +
  expand_limits(y = 8000)

ggplot(lynx_datetime.df, aes(year, lynx)) +
  geom_line() +
  stat_peaks(colour = "red",
    geom = "text_linked",
    position = position_nudge_to(y = 7200),
    arrow = arrow(length = grid::unit(1.5, "mm")),
    hjust = -0.1,
    x.label.fmt = "%Y",
    angle = 90) +
  expand_limits(y = 8000)
Description

`stat_poly_eq` fits a polynomial by default with `stats::lm()` but alternatively using robust or quantile regression. From the fitted model it generates several labels including the equation, p-value, F-value, coefficient of determination (R^2), 'AIC', 'BIC', and number of observations.

Usage

```r
stat_poly_eq(
  mapping = NULL,
  data = NULL,
  geom = "text_npc",
  position = "identity",
  ...
  method = "lm",
  method.args = list(),
  formula = NULL,
  eq.with.lhs = TRUE,
  eq.x.rhs = NULL,
  small.r = FALSE,
  small.p = FALSE,
  coef.digits = 3,
  coef.keep.zeros = TRUE,
  rr.digits = 2,
  f.digits = 3,
  p.digits = 3,
  label.x = "left",
  label.y = "top",
  label.x.npc = NULL,
  label.y.npc = NULL,
  hstep = 0,
  vstep = NULL,
  output.type = NULL,
  na.rm = FALSE,
  orientation = NA,
  parse = NULL,
  show.legend = FALSE,
  inherit.aes = TRUE
)
```

Arguments

`mapping` The aesthetic mapping, usually constructed with `aes` or `aes..`. Only needs to be set at the layer level if you are overriding the plot defaults.
A layer specific dataset, only needed if you want to override the plot defaults.

The geometric object to use display the data

The position adjustment to use for overlapping points on this layer

... other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.

function or character If character, "lm" and "rlm" are accepted. If a function, it must have formal parameters formula and data and return a model fit object for which summary() and coefficients() are consistent with those for lm fits.

named list with additional arguments.

a formula object. Using aesthetic names x and y instead of original variable names.

If character the string is pasted to the front of the equation label before parsing or a logical (see note).

character this string will be used as replacement for "x" in the model equation when generating the label before parsing it.

logical Flags to switch use of lower case r and p for coefficient of determination and p-value.

integer Number of significant digits to use for the fitted coefficients and F-value.

logical Keep or drop trailing zeros when formatting the fitted coefficients and F-value.

integer Number of digits after the decimal point to use for R^2 and P-value in labels.

numeric with range 0..1 "normalized parent coordinates" (npc units) or character if using geom_text_npc() or geom_label_npc(). If using geom_text() or geom_label() numeric in native data units. If too short they will be recycled.

numeric with range 0..1 (npc units) DEPRECATED, use label.x and label.y instead; together with a geom using npcx and npcy aesthetics.

numeric in npc units, the horizontal and vertical step used between labels for different groups.

character One of "expression", "LaTeX", "text", "markdown" or "numeric".

a logical indicating whether NA values should be stripped before the computation proceeds.

character Either "x" or "y" controlling the default for formula.

logical Passed to the geom. If TRUE, the labels will be parsed into expressions and displayed as described in ?plotmath. Default is TRUE if output.type = "expression" and FALSE otherwise.

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.
**stat_poly_eq**

**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.

**Details**

This stat can be used to automatically annotate a plot with $R^2$, adjusted $R^2$ or the fitted model equation. It supports linear regression, robust linear regression and median regression fitted with functions `lm()`, `MASS::rlm()` or `quanreg::rq()`. The $R^2$ and adjusted $R^2$ annotations can be used with any linear model formula. The fitted equation label is correctly generated for polynomials or quasi-polynomials through the origin. Model formulas can use `poly()` or be defined algebraically with terms of powers of increasing magnitude with no missing intermediate terms, except possibly for the intercept indicated by "- 1" or "-1" or "+ 0" in the formula. The validity of the formula is not checked in the current implementation, and for this reason the default aesthetics sets $R^2$ as label for the annotation. This stat generates labels as R expressions by default but LaTeX (use TikZ device), markdown (use package ‘ggtext’) and plain text are also supported, as well as numeric values for user-generated text labels. The value of parse is set automatically based on output-type, but if you assemble labels that need parsing from numeric output, the default needs to be overridden. This stat only generates annotation labels, the predicted values/line need to be added to the plot as a separate layer using `stat_poly_line` or `stat_smooth`, so to make sure that the same model formula is used in all steps it is best to save the formula as an object and supply this object as argument to the different statistics.

A ggplot statistic receives as data a data frame that is not the one passed as argument by the user, but instead a data frame with the variables mapped to aesthetics. `stat_poly_eq()` mimics how `stat_smooth()` works, except that only polynomials can be fitted. Similarly to these statistics the model fits respect grouping, so the scales used for x and y should both be continuous scales rather than discrete.

**Aesthetics**

`stat_poly_eq` understands x and y, to be referenced in the formula and weight passed as argument to parameter weights. All three must be mapped to numeric variables. In addition, the aesthetics understood by the geom ("text" is the default) are understood and grouping respected.

**Computed variables**

If output.type different from "numeric" the returned tibble contains columns listed below. If the model fit function used does not return a value, the label is set to character(0L).

- **x_npcx**: x position
- **y_npcy**: y position
- **eq.label**: equation for the fitted polynomial as a character string to be parsed
- **rr.label**: $R^2$ of the fitted model as a character string to be parsed
- **adj.rr.label**: Adjusted $R^2$ of the fitted model as a character string to be parsed
- **f.value.label**: F value and degrees of freedom for the fitted model as a whole.
- **p.value.label**: P-value for the F-value above.
- **AIC.label**: AIC for the fitted model.
\texttt{BIC.label}  BIC for the fitted model.
\texttt{n.label}  Number of observations used in the fit.
\texttt{grp.label}  Set according to mapping in \texttt{aes}.
\texttt{r.squared, adj.r.squared, p.value, n}  numeric values, from the model fit object

If \texttt{output.type} is "numeric" the returned tibble contains columns listed below. If the model fit function used does not return a value, the variable is set to \texttt{NA_real_}.

\texttt{x.npcx}  \texttt{x} position
\texttt{y.npcy}  \texttt{y} position
\texttt{coef.ls}  list containing the "coefficients" matrix from the summary of the fit object
\texttt{r.squared, adj.r.squared, f.value, f.df1, f.df2, p.value, AIC, BIC, n}  numeric values, from the model fit object
\texttt{grp.label}  Set according to mapping in \texttt{aes}.
\texttt{b_0.constant}  TRUE is polynomial is forced through the origin
\texttt{b_i}  One or columns with the coefficient estimates

To explore the computed values returned for a given input we suggest the use of \texttt{geom_debug} as shown in the last examples below.

\textbf{Note}

For backward compatibility a logical is accepted as argument for \texttt{eq.with.lhs}. If \texttt{TRUE}, the default is used, either \texttt{"x"} or \texttt{"y"}, depending on the argument passed to \texttt{formula}. However, \texttt{"x"} or \texttt{"y"} can be substituted by providing a suitable replacement character string through \texttt{eq.x_rhs}. Parameter orientation is redundant as it only affects the default for \texttt{formula} but is included for consistency with \texttt{ggplot2::stat_smooth()}.  

\textbf{References}

Written as an answer to a question at Stackoverflow. \url{https://stackoverflow.com/questions/7549694/adding-regression-line-equation-and-r2-on-graph}

\textbf{See Also}

This \texttt{stat_poly_eq} statistic can return ready formatted labels depending on the argument passed to \texttt{output.type}. This is possible because only polynomial models are supported. If multiple quantiles are desired, then \texttt{stat_quant_eq} should be used instead of \texttt{stat_poly_eq}. For other types of models such as non-linear models, statistics \texttt{stat_fit_glance} and \texttt{stat_fit_tidy} should be used instead and the code for construction of character strings from numeric values and their mapping to aesthetic \texttt{label} needs to be explicitly supplied in the call.

Other \texttt{ggplot} statistics for linear and polynomial regression: \texttt{stat_poly_line()}
Examples

# generate artificial data
set.seed(4321)
x <- 1:100
y <- (x + x^2 + x^3) + rnorm(length(x), mean = 0, sd = mean(x^3) / 4)
my.data <- data.frame(x = x, y = y,
    group = c("A", "B"),
    y2 = y * c(0.5, 2),
    w = sqrt(x))

# give a name to a formula
formula <- y ~ poly(x, 3, raw = TRUE)

# no weights
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(formula = formula)

# grouping
ggplot(my.data, aes(x, y, color = group)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(formula = formula)

# rotation
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(formula = formula, angle = 90, hjust = 1)

# label location
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(formula = formula, label.y = "bottom", label.x = "right")

ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(formula = formula, label.y = 0.1, label.x = 0.9)

# using weights
ggplot(my.data, aes(x, y, weight = w)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(formula = formula)

# no weights, digits for R square
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(formula = formula, digits = 2)
stat_poly_eq(formula = formula, rr.digits = 4)

# user specified label
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(aes(label = paste(after_stat(rr.label),
                      after_stat(n.label), sep = "\", \"x\")),
                formula = formula)

ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(aes(label = paste(after_stat(eq.label),
                      after_stat(adj.rr.label), sep = "\", \"x\")),
                formula = formula)

ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(aes(label = paste(after_stat(f.value.label),
                      after_stat(p.value.label),
                      sep = "\", \"x\")),
                formula = formula)

# x on y regression
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula, orientation = "y") +
  stat_poly_eq(aes(label = paste(after_stat(eq.label),
                      after_stat(adj.rr.label),
                      sep = "\", \"x\")),
                formula = x ~ poly(y, 3, raw = TRUE))

# conditional user specified label
ggplot(my.data, aes(x, y, color = group)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(aes(label = ifelse(after_stat(adj.r.squared) > 0.96,
                      paste(after_stat(adj.rr.label),
                      after_stat(eq.label),
                      sep = "\", \"x\"),
                      after_stat(adj.rr.label))),
                rr.digits = 3,
                formula = formula)

# geom = "text"
 ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(geom = "text", label.x = 100, label.y = 0, hjust = 1,
                formula = formula)
# using numeric values
# Here we use columns b_0 ... b_3 for the coefficient estimates
my.format <-
  "b[0]="(~%.3g)\$", \"*b[1]="(~%.3g)\$", \"*b[2]="(~%.3g)\$", \"*b[3]="(~%.3g)"

ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(formula = formula,
               output.type = "numeric",
               parse = TRUE,
               mapping =
               aes(label = sprintf(my.format,
                                    after_stat(b_0), after_stat(b_1),
                                    after_stat(b_2), after_stat(b_3))))

# Inspecting the returned data using geom_debug()
if (requireNamespace("gginnards", quietly = TRUE)) {
  library(gginnards)

  # This provides a quick way of finding out the names of the variables that
  # are available for mapping to aesthetics.

  # the whole of data
  ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(formula = formula, geom = "debug")

  # names of the variables
  ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(formula = formula, geom = "debug",
               summary.fun = colnames)

  # only data$eq.label
  ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(formula = formula, geom = "debug",
               output.type = "expression",
               summary.fun = function(x) {x["eq.label"]})

  # only data$eq.label
  ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(aes(label = after_stat(eq.label)),
               output.type = "expression",
               summary.fun = function(x) {x["eq.label"]})
# only data$eq.label

ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(formula = formula, geom = "debug",
               output.type = "latex",
               summary.fun = function(x) {x[['eq.label']]}))

# only data$eq.label

ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(formula = formula, geom = "debug",
               output.type = "text",
               summary.fun = function(x) {x[['eq.label']]}))

# show the content of a list column

ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(formula = formula, geom = "debug",
               output.type = "numeric",
               summary.fun = function(x) {x[['coef.ls']]$[1]})
}

stat_poly_line

Predicted line from model fit

Description

Predicted values and a confidence band are computed and, by default, plotted. stat_poly_line() behaves like stat_smooth except for supporting the use of y as explanatory variable in the model formula, fits the model with stats::lm() as default for method, irrespective of the number of observations. The fit can alternatively by done by any of the methods supported by stat_smooth, including method = "auto".

Usage

stat_poly_line(
  mapping = NULL,
  data = NULL,
  geom = "smooth",
  position = "identity",
  ...,  
  method = "lm",
  formula = formula, geom = "debug",
  output.type = "markdown",
  summary.fun = function(x) {x[['eq.label']]}))
formula = NULL,
se = TRUE,
n = 80,
span = 0.75,
fullrange = FALSE,
level = 0.95,
method.args = list(),
na.rm = FALSE,
orientation = NA,
show.legend = NA,
inherit.aes = TRUE
}

Arguments

mapping The aesthetic mapping, usually constructed with `aes` or `aes_args`. Only needs to be
set at the layer level if you are overriding the plot defaults.
data A layer specific dataset, only needed if you want to override the plot defaults.
geom The geometric object to use display the data
position The position adjustment to use for overlapping points on this layer
... other arguments passed on to `layer`. This can include aesthetics whose values
you want to set, not map. See `layer` for more details.
method function or character If character, "lm", "rlm" and "rq" are accepted. If a func-
tion, it must have formal parameters `formula` and `data` and return a model fit
object for which `summary()` and `coefficients()` are consistent with those for
`lm` fits.
formula a formula object. Using aesthetic names `x` and `y` instead of original variable
names.
se Display confidence interval around smooth? (‘TRUE’ by default, see ‘level’ to
control.)
n Number of points at which to evaluate smoother.
span Controls the amount of smoothing for the default loess smoother. Smaller num-
ers produce wigglier lines, larger numbers produce smoother lines. Only used
with `loess`, i.e. when `method = "loess"`; or when `method = NULL` (the de-
fault) and there are fewer than 1,000 observations.
fullrange Should the fit span the full range of the plot, or just the data?
level Level of confidence interval to use (0.95 by default).
method.args named list with additional arguments.
na.rm a logical indicating whether NA values should be stripped before the compu-
tation proceeds.
orientation character Either "x" or "y" controlling the default for `formula`.
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.
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`.
Details

This statistic is just `stat_smooth` with different defaults and updated so that it interprets the argument passed to `formula` differently, accepting `y` as explanatory variable and setting `orientation` automatically. In addition the default for `method` is "lm", matching the default used in `stat_poly_eq()` and `stat_poly_quant()`. It calls `StatSmooth` to build a layer.

`geom_smooth`, which is used by default, treats each axis differently and can thus have two orientations. The orientation is easy to deduce from the argument passed to `formula`. Thus, `stat_smooth_xy()` will by default guess which orientation the layer should have. If no argument is passed to `formula`, the orientation is ambiguous. In that case the orientation can be specified directly passing an argument to the `orientation` parameter, which can be either "x" or "y". The value gives the axis that is taken as the explanatory variable, "x" being the default orientation you would expect for the geom. Package 'ggpmisc' does not define new geometries matching the new statistics as they are not needed and conceptually transformations of data are expressed as statistics.

Value

The value returned by the statistic is a data frame, that will have `n` rows of predicted values and and their confidence limits.

Computed variables

`'stat_smooth_xy()'` provides the following variables, some of which depend on the orientation:

- `y *or* x` predicted value
- `ymin *or* xmin` lower pointwise confidence interval around the mean
- `ymax *or* xmax` upper pointwise confidence interval around the mean
- `se` standard error

Aesthetics

`stat_poly_eq` understands `x` and `y`, to be referenced in the `formula` and `weight` passed as argument to parameter weights. All three must be mapped to numeric variables. In addition, the aesthetics understood by the geom ("geom_smooth" is the default) are understood and grouping respected.

See Also

Other ggplot statistics for linear and polynomial regression: `stat_poly_eq()`

Examples

```r
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  stat_poly_line()

ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  stat_poly_line(formula = x ~ y)

ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  stat_poly_line(formula = y ~ x)
```

```r
geom_point() +
stat_poly_line(formula = y ~ poly(x, 3))
```

```r
ggplot(mpg, aes(displ, hwy)) +
geom_point() +
stat_poly_line(formula = x ~ poly(y, 3))
```

# The default behavior of geom_smooth()
```r
ggplot(mpg, aes(displ, hwy)) +
geom_point() +
stat_poly_line(method = "auto")
```

# Use span to control the "wiggliness" of the default loess smoother.
# The span is the fraction of points used to fit each local regression:
# small numbers make a wigglier curve, larger numbers make a smoother curve.
```r
ggplot(mpg, aes(displ, hwy)) +
geom_point() +
stat_poly_line(method = "loess", span = 0.3)
```

```r
ggplot(mpg, aes(displ, hwy)) +
geom_point() +
stat_poly_line(method = lm, formula = y ~ splines::bs(x, 3), se = FALSE)
```

```r
ggplot(mpg, aes(displ, hwy)) +
geom_point() +
stat_poly_line(method = lm, formula = x ~ splines::bs(y, 3), se = FALSE)
```

# Smooths are automatically fit to each group (defined by categorical
# aesthetics or the group aesthetic) and for each facet.
```r
ggplot(mpg, aes(displ, hwy, colour = class)) +
geom_point() +
stat_poly_line(se = FALSE)
```

```r
ggplot(mpg, aes(displ, hwy)) +
geom_point() +
stat_poly_line(method = "auto", span = 0.8) +
facet_wrap(~drv)
```

---

**stat_quant_band**

Compute predicted line from quantile regression fit

---

**Description**

Predicted values are computed and, by default, plotted as a band plus an optional line within.

`stat_quant_band()` supports the use of both x and y as explanatory variable in the model formula.
Usage

```r
calling_sequence

stat_quant_band(
  mapping = NULL,
  data = NULL,
  geom = "smooth",
  position = "identity",
  ..., 
  quantiles = c(0.25, 0.5, 0.75),
  formula = NULL,
  n = 80,
  method = "rq",
  method.args = list(),
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

- **mapping**: The aesthetic mapping, usually constructed with `aes` or `aes_`. Only needs to be set at the layer level if you are overriding the plot defaults.
- **data**: A layer specific dataset, only needed if you want to override the plot defaults.
- **geom**: The geometric object to use display the data.
- **position**: The position adjustment to use for overlapping points on this layer.
- **quantiles**: numeric vector Two or three values in 0..1 indicating the quantiles at the edges of the band and optionally a line within the band.
- **formula**: a formula object. Using aesthetic names `x` and `y` instead of original variable names.
- **n**: Number of points at which to evaluate smoother.
- **method**: function or character If character, "lm", "rlm" and "rq" are accepted. If a function, it must have formal parameters `formula` and `data` and return a model fit object for which `summary()` and `coefficients()` are consistent with those for `lm` fits.
- **method.args**: named list with additional arguments.
- **na.rm**: a logical indicating whether NA values should be stripped before the computation proceeds.
- **orientation**: character Either "x" or "y" controlling the default for `formula`.
- **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.
- **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`. 
Details
This statistic is similar to \texttt{stat\_quant\_line} but plots the quantiles differently with the band representing a region between two quantiles, while in \texttt{stat\_quant\_line()} the bands plotted when \texttt{se = TRUE} represent confidence intervals for the fitted quantile lines.

\texttt{geom\_smooth}, which is used by default, treats each axis differently and thus is dependent on orientation. If no argument is passed to \texttt{formula}, it defaults to \texttt{y ~ x} but \texttt{x ~ y} is also accepted, and equivalent to \texttt{y ~ x} plus \texttt{orientation = "y"}. Package 'ggpmisc' does not define a new geometry matching this statistic as it is enough for the statistic to return suitable 'x' and 'y' values.

Value
The value returned by the statistic is a data frame, that will have \( n \) rows of predicted values for three quantiles as \( y, y_{\text{min}} \) and \( y_{\text{max}} \), plus \( x \).

Aesthetics
\texttt{stat\_quant\_eq} expects \( x \) and \( y \), aesthetics to be used in the \texttt{formula} rather than the names of the variables mapped to them. If present, the variable mapped to the weight aesthetics is passed as argument to parameter \texttt{weights} of the fitting function. All three must be mapped to numeric variables. In addition, the aesthetics recognized by the geometry ("\texttt{geom\_smooth}" is the default) are obeyed and grouping respected.

Note
Even though the \texttt{ggplot2::geom\_smooth()} is used by default, the default colour of the band is different to avoid confusion of between quantile bands and confidence bands. Of course, the \texttt{fill} aesthetics can be mapped to a variable or to a constant as usual (\texttt{fill = "grey60"} restores the geom’s original default).

See Also
Other \texttt{ggplot} statistics for quantile regression: \texttt{stat\_quant\_eq()}, \texttt{stat\_quant\_line}()

Examples
```r
# ggplot2::geom_smooth() is used by default, default color of the band is different to avoid confusion of between quantile bands and confidence bands. Of course, the fill aesthetics can be mapped to a variable or to a constant as usual (fill = "grey60" restores the geom’s original default).

ggplot(mpg, aes(displ, hwy)) + geom_point() + stat_quant_band()

# If you need the fitting to be done along the y-axis set the orientation
ggplot(mpg, aes(displ, hwy)) + geom_point() + stat_quant_band(orientation = "y")

# ggplot2::geom_smooth() is used by default, default color of the band is different to avoid confusion of between quantile bands and confidence bands. Of course, the fill aesthetics can be mapped to a variable or to a constant as usual (fill = "grey60" restores the geom’s original default).

ggplot(mpg, aes(displ, hwy)) + geom_point() + stat_quant_band(formula = y ~ x)

ggplot(mpg, aes(displ, hwy)) + geom_point() +
stat_quant_band(formula = x ~ y)

ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  stat_quant_band(formula = y ~ poly(x, 3))

ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  stat_quant_band(formula = x ~ poly(y, 3))

# Instead of rq() we can use rqss() to fit an additive model:
library(quantreg)

ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  stat_quant_band(method = "rqss",
                  formula = y ~ qss(x))

ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  stat_quant_band(method = "rqss",
                  formula = x ~ qss(y, constraint = "D"))

# Regressions are automatically fit to each group (defined by categorical
# aesthetics or the group aesthetic) and for each facet.

ggplot(mpg, aes(displ, hwy, colour = class)) +
  geom_point() +
  stat_quant_band(formula = y ~ x)

ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  stat_quant_band(formula = y ~ poly(x, 2)) +
  facet_wrap(~drv)

ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  stat_quant_band(linetype = "dashed", color = "darkred", fill = "red")

ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  stat_quant_band(color = NA, alpha = 1)

ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  stat_quant_band(quantiles = c(0, 0.1, 0.2))

stat_quant_eq  
Equation, p-value, R^2, AIC or BIC from quantile regression
stat_quant_eq

Description

stat_quant_eq fits a polynomial model by quantile regression and generates several labels including the equation, p-value, coefficient of determination (R^2), 'AIC' and 'BIC'.

Usage

stat_quant_eq(
  mapping = NULL,
  data = NULL,
  geom = "text_npc",
  position = "identity",
  ..., 
  formula = NULL,
  quantiles = c(0.25, 0.5, 0.75),
  eq.with.lhs = TRUE,
  eq.x.rhs = NULL,
  coef.digits = 3,
  coef.keep.zeros = TRUE,
  rho.digits = 2,
  label.x = "left",
  label.y = "top",
  label.x.npc = NULL,
  label.y.npc = NULL,
  hstep = 0,
  vstep = NULL,
  output.type = "expression",
  na.rm = FALSE,
  orientation = NA,
  parse = NULL,
  show.legend = FALSE,
  inherit.aes = TRUE
)

Arguments

mapping The aesthetic mapping, usually constructed with aes or aes_. Only needs to be set at the layer level if you are overriding the plot defaults.
data A layer specific dataset, only needed if you want to override the plot defaults.
geom The geometric object to use display the data
position The position adjustment to use for overlapping points on this layer
... other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
formula a formula object. Using aesthetic names instead of original variable names.
quantiles numeric vector Values in 0..1 indicating the quantiles.
eq.with.lhs If character the string is pasted to the front of the equation label before parsing or a logical (see note).
eq.x.rhs character this string will be used as replacement for "x" in the model equation when generating the label before parsing it.

coeff.digits, rho.digits integer Number of significant digits to use for the fitted coefficients and rho in labels.

coeff.keep.zeros logical Keep or drop trailing zeros when formatting the fitted coefficients and F-value.

label.x, label.y numeric with range 0..1 "normalized parent coordinates" (npc units) or character if using geom_text_npc() or geom_label_npc(). If using geom_text() or geom_label() numeric in native data units. If too short they will be recycled.

label.x.npc, label.y.npc numeric with range 0..1 (npc units) DEPRECATED, use label.x and label.y instead; together with a geom using npcx and npcy aesthetics.

hstep, vstep numeric in npc units, the horizontal and vertical step used between labels for different groups.

output.type character One of "expression", "LaTeX", "text", "markdown" or "numeric". In most cases, instead of using this statistics to obtain numeric values, it is better to use stat_fit_tidy().

na.rm a logical indicating whether NA values should be stripped before the computation proceeds.

orientation character Either "x" or "y" controlling the default for formula.

parse logical Passed to the geom. If TRUE, the labels will be parsed into expressions and displayed as described in ?plotmath. Default is TRUE if output.type = "expression" and FALSE otherwise.

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.

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.

Details

This statistic interprets the argument passed to formula differently than stat_quantile accepting y as well as x as explanatory variable, matching stat_poly_quant().

When two variables are subject to mutual constrains, it is useful to consider both of them as explanatory and interpret the relationship based on them. So, from version 0.4.1 'ggpmisc' makes it possible to easily implement the approach described by Cardoso (2019) under the name of "Double quantile regression".

This stat can be used to automatically annotate a plot with R^2, adjusted R^2 or the fitted model equation. It supports only linear models fitted with function lm(). The R^2 and adjusted R^2 annotations can be used with any linear model formula. The fitted equation label is correctly generated for polynomials or quasi-polynomials through the origin. Model formulas can use poly() or be defined algebraically with terms of powers of increasing magnitude with no missing intermediate
terms, except possibly for the intercept indicated by "-1" or "-1" or "+ 0" in the formula. The validity of the formula is not checked in the current implementation, and for this reason the default aesthetics sets R^2 as label for the annotation. This stat generates labels as R expressions by default but LaTeX (use TikZ device), markdown (use package 'ggtext') and plain text are also supported, as well as numeric values for user-generated text labels. The value of parse is set automatically based on output-type, but if you assemble labels that need parsing from numeric output, the default needs to be overridden. This stat only generates annotation labels, the predicted values/line need to be added to the plot as a separate layer using `stat_quant_line` or `stat_quantile`, so to make sure that the same model formula is used in all steps it is best to save the formula as an object and supply this object as argument to the different statistics.

A ggplot statistic receives as data a data frame that is not the one passed as argument by the user, but instead a data frame with the variables mapped to aesthetics. `stat_quant_eq()` mimics how `stat_smooth()` works, except that only polynomials can be fitted. In other words, it respects the grammar of graphics. This helps ensure that the model is fitted to the same data as plotted in other layers.

### Aesthetics

`stat_quant_eq` understands x and y, to be referenced in the formula and weight passed as argument to parameter weights of `lm()`. All three must be mapped to numeric variables. In addition, the aesthetics understood by the geom used ("text" by default) are understood and grouping respected.

### Computed variables

If output.type different from "numeric" the returned tibble contains columns below in addition to a modified version of the original group:

- **x.npcx** x position
- **y.npcy** y position
- **coef.ls, rsquared, adj.rsquared, AIC, BIC** as numeric values extracted from fit object
- **eq.label** equation for the fitted polynomial as a character string to be parsed
- **rho.label** rho of the fitted model as a character string to be parsed
- **AIC.label** AIC for the fitted model.
- **n.label** Number of observations used in the fit.
- **rq.method** character, method used.
- **rho, n** numeric values extracted or computed from fit object.
- **hjust, vjust** Set to "inward" to override the default of the "text" geom.
- **quantile** Numeric value of the quantile used for the fit
- **quantile.f** Factor with a level for each quantile

If output.type is "numeric" the returned tibble contains columns in addition to a modified version of the original group:

- **x.npcx** x position
- **y.npcy** y position
**coef.ls** list containing the "coefficients" matrix from the summary of the fit object

**rho, AIC, n** numeric values extracted or computed from fit object

**rq.method** character, method used.

**hjust, vjust** Set to "inward" to override the default of the "text" geom.

**quantile** Indicating the quantile used for the fit

**quantile.f** Factor with a level for each quantile

**b_0.constant** TRUE is polynomial is forced through the origin

**b_i** One or columns with the coefficient estimates

To explore the computed values returned for a given input we suggest the use of `geom_debug` as shown in the example below.

**Note**

For backward compatibility a logical is accepted as argument for `eq.with.lhs`. If TRUE, the default is used, either "x" or "y", depending on the argument passed to `formula`. However, "x" or "y" can be substituted by providing a suitable replacement character string through `eq.x.rhs`. Parameter orientation is redundant as it only affects the default for `formula` but is included for consistency with `ggplot2::stat_smooth()`.

Support for the angle aesthetic is not automatic and requires that the user passes as argument suitable numeric values to override the defaults for label positions.

**References**


**See Also**

This `stat_quant_eq` statistic can return ready formatted labels depending on the argument passed to `output.type`. This is possible because only polynomial models are supported. For other types of models, statistics `stat_fit_glance`, `stat_fit_tidy` and `stat_fit_glance` should be used instead and the code for construction of character strings from numeric values and their mapping to aesthetic label needs to be explicitly supplied in the call.

Other ggplot statistics for quantile regression: `stat_quant_band()`, `stat_quant_line()`

**Examples**

```r
# generate artificial data
set.seed(4321)
x <- 1:100
y <- (x + x^2 + x^3) + rnorm(length(x), mean = 0, sd = mean(x^3) / 4)
my.data <- data.frame(x = x, y = y,
                      group = c("A", "B"),
y2 = y * c(0.5, 2),
w = sqrt(x))

# using defaults
```
```r
# same formula as default
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(formula = y ~ x) +
  stat_quant_eq(formula = y ~ x)

# explicit formula "x explained by y"
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(formula = x ~ y) +
  stat_quant_eq(formula = x ~ y)

# using color
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(aes(color = after_stat(quantile.f))) +
  stat_quant_eq(aes(color = after_stat(quantile.f))) +
  labs(color = "Quantiles")

# location and colour
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(aes(color = after_stat(quantile.f))) +
  stat_quant_eq(aes(color = after_stat(quantile.f)),
               label.y = "bottom", label.x = "right") +
  labs(color = "Quantiles")

# give a name to a formula
formula <- y ~ poly(x, 3, raw = TRUE)

# no weights
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(formula = formula) +
  stat_quant_eq(formula = formula)

# angle
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(formula = formula) +
  stat_quant_eq(formula = formula, angle = 90, hstep = 0.05, vstep = 0,
               label.y = 0.98, hjust = 1)

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(formula = formula) +
  stat_quant_eq(formula = formula, angle = 90,
               hstep = 0.05, vstep = 0, hjust = 0,
               vjust = 0, stack = FALSE)
```
# user set quantiles
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(formula = formula, quantiles = 0.5) +
  stat_quant_eq(formula = formula, quantiles = 0.5)

# grouping
ggplot(my.data, aes(x, y, color = group)) +
  geom_point() +
  stat_quant_line(formula = formula) +
  stat_quant_eq(formula = formula)

ggplot(my.data, aes(x, y, color = group)) +
  geom_point() +
  stat_quant_line(formula = formula) +
  stat_quant_eq(formula = formula, angle = 90,
               hstep = 0.05, vstep = 0, hjust = 0,
               size = 3, label.y = 0.3)

# labelling equations
ggplot(my.data, aes(x, y, shape = group, linetype = group,
                    grp.label = group)) +
  geom_point() +
  stat_quant_line(formula = formula, color = "black") +
  stat_quant_eq(aes(label = paste(after_stat(grp.label), after_stat(eq.label), sep = "\": ", label.y = 0.3))

# setting non-default quantiles
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(formula = formula,
                  quantiles = c(0.1, 0.5, 0.9)) +
  stat_quant_eq(formula = formula, parse = TRUE,
                quantiles = c(0.1, 0.5, 0.9))

# Location of equations
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(formula = formula) +
  stat_quant_eq(formula = formula, label.y = "bottom", label.x = "right")

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(formula = formula) +
  stat_quant_eq(formula = formula, label.y = 0.03, label.x = 0.95, vstep = 0.04)

# using weights
ggplot(my.data, aes(x, y, weight = w)) +
  geom_point() +
  stat_quant_line(formula = formula) +
  stat_quant_eq(formula = formula) +
  geom_point() +
  stat_quant_line(formula = formula) +
  stat_quant_eq(formula = formula) +
  geom_point() +
  stat_quant_line(formula = formula) +
  stat_quant_eq(formula = formula) +
  geom_point() +
  stat_quant_line(formula = formula) +
  stat_quant_eq(formula = formula) +
stat_quant_eq

stat_quant_eq(formula = formula)

# no weights, quantile set to upper boundary
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(formula = formula, quantiles = 0.95) +
  stat_quant_eq(formula = formula, quantiles = 0.95)

# user specified label
ggplot(my.data, aes(x, y, color = group, grp.label = group)) +
  geom_point() +
  stat_quant_line(method = "rq", formula = formula,
                  quantiles = c(0.05, 0.5, 0.95)) +
  stat_quant_eq(aes(label = paste(after_stat(grp.label), "x": "x",
                      after_stat(eq.label), sep = ""),
                  quantiles = c(0.05, 0.5, 0.95),
                  formula = formula, size = 3))

# geom = "text"
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(method = "rq", formula = formula, quantiles = 0.5) +
  stat_quant_eq(label.x = "left", label.y = "top",
                formula = formula)

# Inspecting the returned data using geom_debug()
if (requireNamespace("gginnards", quietly = TRUE)) {
  library(gginnards)

  # This provides a quick way of finding out the names of the variables that
  # are available for mapping to aesthetics.
  ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_eq(formula = formula, geom = "debug")

  ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_eq(aes(label = after_stat(eq.label)),
                formula = formula, geom = "debug",
                output.type = "markdown")

  ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_eq(formula = formula, geom = "debug", output.type = "text")

  ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_eq(formula = formula, geom = "debug", output.type = "numeric")

  ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_eq(formula = formula, quantiles = c(0.25, 0.5, 0.75),
stat_quant_line

Predicted line from quantile regression fit

Description

Predicted values are computed and, by default, plotted. Depending on the fit method, a confidence band can be computed and plotted. The confidence band can be interpreted similarly as that produced by stat_smooth() and stat_poly_line().

Usage

```r
stat_quant_line(
  mapping = NULL,
  data = NULL,
  geom = "smooth",
  position = "identity",
  ..., 
  quantiles = c(0.25, 0.5, 0.75),
  formula = NULL,
  se = length(quantiles) == 1L,
  n = 80,
  method = "rq",
  method.args = list(),
  level = 0.95,
  type = "direct",
  interval = "confidence",
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

- **mapping**: The aesthetic mapping, usually constructed with `aes` or `aes_`. Only needs to be set at the layer level if you are overriding the plot defaults.
- **data**: A layer specific dataset, only needed if you want to override the plot defaults.
- **geom**: The geometric object to use display the data
**position**
The position adjustment to use for overlapping points on this layer

... other arguments passed on to `layer`. This can include aesthetics whose values you want to set, not map. See `layer` for more details.

**quantiles**
numeric vector Values in 0..1 indicating the quantiles.

**formula**
a formula object. Using aesthetic names `x` and `y` instead of original variable names.

**se**
logical Passed to `quantreg::predict.rq()`.

**n**
Number of points at which to evaluate smoother.

**method**
function or character If character, "rq" and "rqss" are accepted. If a function, it must have formal parameters `formula` and `data` and return a model fit object for which `summary()` and `coefficients()` are consistent with those for `lm` fits.

**method.args**
named list with additional arguments passed to `rq()` or `rqss()`.

**level**
umeric in range [0..1] Passed to `quantreg::predict.rq()`.

**type**
character Passed to `quantreg::predict.rq()`.

**interval**
character Passed to `quantreg::predict.rq()`.

**na.rm**
a logical indicating whether NA values should be stripped before the computation proceeds.

**orientation**
character Either "x" or "y" controlling the default for `formula`.

**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.

**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`.

### Details

`stat_quant_line()` behaves similarly to `ggplot2::stat_smooth()` and `stat_poly_line()` but supports fitting regressions for multiple quantiles in the same plot layer. This statistic interprets the argument passed to `formula` accepting `y` as well as `x` as explanatory variable, matching `stat_quant_eq()`.

While `stat_quant_eq()` supports only method "rq", `stat_quant_line()` and `stat_quant_band()` support both "rq" and "rqss". In the case of "rqss" the model formula makes normally use of `qss()` to formulate the spline and its constraints.

`geom_smooth`, which is used by default, treats each axis different and thus is dependent on orientation. If no argument is passed to `formula`, it defaults to `y ~ x`. Formulas with `y` as explanatory variable are treated as if `x` was the explanatory variable and `orientation = "y"`.

Package 'ggpmisc' does not define a new geometry matching this statistic as it is enough for the statistic to return suitable `x`, `y`, `ymin`, `ymax` and `group` values.

There are multiple uses for double regression on `x` and `y`. For example, when two variables are subject to mutual constrains, it is useful to consider both of them as explanatory and interpret the relationship based on them. So, from version 0.4.1 'ggpmisc' makes it possible to easily implement the approach described by Cardoso (2019) under the name of "Double quantile regression".
Value

The value returned by the statistic is a data frame, that will have \( n \) rows of predicted values and and their confidence limits for each quantile, with each quantile in a group. The variables are \( x \) and \( y \) with \( y \) containing predicted values. In addition, quantile and quantile.f indicate the quantile used and and edited group preserves the original grouping adding a new "level" for each quantile. If \( se = \text{TRUE} \), a confidence band is computed and values for it returned in \( y_{\text{max}} \) and \( y_{\text{min}} \).

Aesthetics

\texttt{stat_quant_line} understands \( x \) and \( y \), to be referenced in the formula and weight passed as argument to parameter weights. All three must be mapped to numeric variables. In addition, the aesthetics understood by the geom ("geom_smooth" is the default) are understood and grouping respected.

References


See Also

\texttt{rq, rqss} and \texttt{qss}.

Other ggplot statistics for quantile regression: \texttt{stat_quant_band()}, \texttt{stat_quant_eq()}

Examples

```r
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  stat_quant_line()

ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  stat_quant_line(se = \text{TRUE})

# If you need the fitting to be done along the y-axis set the orientation
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  stat_quant_line(orientation = "y")

ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  stat_quant_line(orientation = "y", se = \text{TRUE})

ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  stat_quant_line(formula = y \sim x)

ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  stat_quant_line(formula = x \sim y)
```
Swap x and y in a formula
symmetric_limits

Description
By default a formula of x on y is converted into a formula of y on x, while the reverse swap is done only if backward = TRUE.

Usage
swap_xy(f, backwards = FALSE)

Arguments
f formula An R model formula
backwards logical

Details
This function is meant to be used only as a helper within ‘ggplot2’ statistics. Normally together with geometries supporting orientation when we want to automate the change in orientation based on a user-supplied formula. Only x and y are changed, and in other respects the formula is rebuilt copying the environment from f.

Value
A copy of f with x and y swapped by each other in the lhs and rhs.

symmetric_limits expand a range to make it symmetric

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
x numeric The automatic limits when used as argument to a scale’s limits formal parameter. Otherwise a numeric vector, possibly a range, for which to compute a symmetric 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.
Examples

symmetric_limits(c(-1, 1.8))
symmetric_limits(c(-10, 1.8))
symmetric_limits(-5:20)

---

volcano_example.df  Example gene expression data

Description

A dataset containing reshaped and simplified output from an analysis of data from RNAseq done with package edgeR. Original data from gene expression in the plant species *Arabidopsis thaliana*.

Usage

volcano_example.df

Format

A data.frame object with 1218 rows and 5 variables

See Also

Other Transcriptomics data examples: quadrant_example.df

Examples

colnames(volcano_example.df)
head(volcano_example.df)

---

xy_outcomes2factor  Convert two numeric ternary outcomes into a factor

Description

Convert two numeric ternary outcomes into a factor

Usage

xy_outcomes2factor(x, y)

xy_thresholds2factor(x, y, x_threshold = 0, y_threshold = 0)
Arguments

x, y
numeric vectors of -1, 0, and +1 values, indicating down regulation, uncertain response or up-regulation, or numeric vectors that can be converted into such values using a pair of thresholds.

x_threshold, y_threshold
numeric vector Ranges enclosing the values to be considered uncertain for each of the two vectors.

Details

This function converts the numerically encoded values into a factor with the four levels "xy", "x", "y" and "none". The factor created can be used for faceting or can be mapped to aesthetics.

Note

This is an utility function that only saves some typing. The same result can be achieved by a direct call to factor. This function aims at making it easier to draw quadrant plots with facets based on the combined outcomes.

See Also

Other Functions for quadrant and volcano plots: FC_format(), outcome2factor(), scale_colour_outcome(), scale_shape_outcome(), scale_y_Pvalue()
Other scales for omics data: outcome2factor(), scale_shape_outcome(), scale_x_logFC()

Examples

xy_outcomes2factor(c(-1, 0, 1), c(0, 1, -1))
xy_thresholds2factor(c(-1, 0, 1), c(0, 1, -1))
xy_thresholds2factor(c(-1, 0, 0.1, -5), c(0, 2, 0, 1, -1))
Index

* Functions for quadrant and volcano plots
  outcome2factor, 5
  scale_colour_outcome, 7
  scale_shape_outcome, 9
  scale_y_Pvalue, 13
  xy_outcomes2factor, 69

* Transcriptomics data examples
  quadrant_example.df, 6
  volcano_example.df, 69

* datasets
  quadrant_example.df, 6
  volcano_example.df, 69

* ggplot statistics for linear and polynomial regression
  stat_poly_eq, 43
  stat_poly_line, 50

* ggplot statistics for model fits
  stat_fit_augment, 15
  stat_fit_deviations, 19
  stat_fit_glance, 22
  stat_fit_residuals, 26
  stat_fit_tb, 30
  stat_fit_tidy, 35

* ggplot statistics for quantile regression
  stat_quant_band, 53
  stat_quant_eq, 56
  stat_quant_line, 64

* scales for omics data
  outcome2factor, 5
  scale_shape_outcome, 9
  scale_x_logFC, 10
  xy_outcomes2factor, 69

aes, 16, 19, 23, 27, 30, 35, 40, 43, 51, 54, 57, 64
aes_, 16, 19, 23, 27, 30, 35, 40, 43, 51, 54, 57, 64
append_layers (Moved), 5
borders, 16, 20, 23, 27, 31, 36, 41, 45, 51, 54,

scale_color_outcome (scale_colour_outcome), 7
scale_colour_outcome, 6, 7, 9, 15, 70
scale_continuous, 12, 15
scale_fill_outcome, 6
scale_fill_outcome (scale_colour_outcome), 7
scale_manual, 8, 9
scale_shape_outcome, 6, 8, 9, 12, 15, 70
scale_x_FDR (scale_y_Pvalue), 13
scale_x_logFC, 6, 9, 10, 70
scale_x_Pvalue (scale_y_Pvalue), 13
scale_y_FDR (scale_y_Pvalue), 13
scale_y_logFC (scale_x_logFC), 10
scale_y_Pvalue, 6, 8, 9, 13, 70
shift_layers (Moved), 5
sprintf, 40
stat_debug_group, 5
stat_debug_group (Moved), 5
stat_debug_panel, 5
stat_debug_panel (Moved), 5
stat_fit_augment, 15, 20, 24, 25, 28, 32, 36, 37
stat_fit_deviations, 17, 19, 25, 28, 32, 37
stat_fit_glance, 16, 17, 20, 22, 28, 32, 36, 37, 46, 60
stat_fit_residuals, 17, 20, 25, 26, 32, 37
stat_fit_tb, 17, 20, 25, 28, 30, 35, 37
stat_fit_tidy, 16, 17, 20, 24, 25, 28, 32, 35, 46, 60
stat_peaks, 39
stat_poly_eq, 16, 24, 36, 43, 52
stat_poly_line, 45, 46, 50
stat_quant_band, 53, 60, 66
stat_quant_eq, 46, 55, 56, 66
stat_quant_line, 55, 59, 60, 64
stat_quantile, 58, 59
stat_smooth, 45, 50
stat_smooth (stat_peaks), 39
StatSmooth, 52
strftime, 40
strptime, 40
swap_xy, 67
symmetric_limits, 68

threshold2factor (outcome2factor), 5
top_layer (Moved), 5
ttheme_gtdefault, 32

volcano_example.df, 7, 69