Package ‘ggpmisc’

December 2, 2019

Type  Package
Title  Miscellaneous Extensions to 'ggplot2'
Version  0.3.3
Date  2019-11-23
Maintainer  Pedro J. Aphalo <pedro.aphalo@helsinki.fi>
Description  Extensions to 'ggplot2' respecting the grammar of graphics paradigm. Specialization of method ggplot(): accept and convert on the fly time series data. Geom: ``table'', ``plot'' and ``grob'' add insets to plots using native data coordinates, while ``table_npc'', ``plot_npc'' and ``grob_npc'' do the same using ``npc'' coordinates through new aesthetics ``npcx'' and ``npcy''. Statistics: locate and tag peaks and valleys; count observations in different quadrants of a plot; select observations based on 2D density; label 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' 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 (>= 3.5.0), ggplot2 (>= 3.2.1)
Imports  grid, rlang (>= 0.4.0), magrittr (>= 1.5), gridExtra (>= 2.3), scales (>= 1.0.0), MASS (>= 7.3-51.3), polynom (>= 1.4-0), splust2R (>= 1.2-2), tibble (>= 2.1.3), plyr (>= 1.8.4), dplyr (>= 0.8.3), xts (>= 0.11-2), zoo (>= 1.8-6), broom (>= 0.5.2), lubridate (>= 1.7.4), stringr (>= 1.4.0)
Suggests  knitr (>= 1.25), rmarkdown (>= 1.15), nlme (>= 3.1-140), gginnards(>= 0.0.2), ggrepel (>= 0.8.1), magick (>= 2.2)
URL  https://www.r4photobiology.info,
      https://bitbucket.org/aphalo/ggpmisc
BugReports https://bitbucket.org/aphalo/ggpmisc/issues
Encoding UTF-8
RoxygenNote 7.0.1
VignetteBuilder knitr
NeedsCompilation no
Author Pedro J. Aphalo [aut, cre] (<https://orcid.org/0000-0003-3385-972X>),
Kamil Slowikowski [ctb]
Repository CRAN
Date/Publication 2019-12-01 23:00:21 UTC

R topics documented:

- ggpmisc-package
- geom_grob
- geom_label_npc
- geom_plot
- geom_quadrant_lines
- geom_table
- geom_x_margin_lines
- geom_x_margin_point
- ggplot
- Moved
- outcome2factor
- quadrant_example.df
- scale_colour_outcome
- scale_continuous_npc
- scale_shape_outcome
- scale_x_logFC
- scale_y_Pvalue
- stat_apply_group
- stat_dens2d_filter
- stat_dens2d_labels
- stat_fit_augment
- stat_fit_deviations
- stat_fit_glance
- stat_fit_residuals
- stat_fit_tb
- stat_fit_tidy
- stat_fnt_tb
- stat_peaks
- stat_poly_eq
- stat_quadrant_counts
- symmetric_limits
- try_data_frame
Extensions to 'ggplot2' respecting the grammar of graphics paradigm. Specialization of method `ggplot()`: accept and convert on the fly time series data. Geom: "table", "plot" and "grob" add insets to plots using native data coordinates, while "table_npc", "plot_npc" and "grob_npc" do the same using "npc" coordinates through new aesthetics "npcx" and "npcy". Statistics: locate and tag peaks and valleys; count observations in different quadrants of a plot; select observations based on 2D density; label 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' 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 [contributor]

References

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

See Also

Useful links:

• https://www.r4photobiology.info
• https://bitbucket.org/aphalo/ggpmisc
• Report bugs at https://bitbucket.org/aphalo/ggpmisc/issues

Examples

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 = ..eq.label..), formula = formula,
                parse = TRUE)

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") +
  theme_classic()

---

gem_grob

Inset graphical objects

description

gem_grob adds a Grob as inset to the ggplot using syntax similar to that of gem_label.

usage

gem_grob(    
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,    
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = FALSE
)

gem_grob_npc(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,    
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = FALSE
)
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.
- **stat**: The statistical transformation to use on the data for this layer, as a string.
- **position**: Position adjustment, either as a string, or the result of a call to a position adjustment function.
- **...**: other arguments passed on to `layer`. This can include aesthetics whose values you want to set, not map. See `layer` for more details.
- **na.rm**: If `FALSE` (the default), removes missing values with a warning. If `TRUE` silently removes missing values.
- **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

Note the "width" and "height" like of a text element are 0, so stacking and dodging Grobs will not work by default, and axis limits are not automatically expanded to include all inset Grobs. Obviously, Grobs do have height and width, but they are in physical units, not data units. The amount of space they occupy on the main plot is constant in data units.

Alignment

You can modify table alignment with the `vjust` and `hjust` aesthetics. These can either be a number between 0 (right/bottom) and 1 (top/left) or a character ("left", "middle", "right", "bottom", "center", "top").

Inset size

You can modify inset plot size with the `vp.width` and `vp.height` aesthetics. These can be a number between 0 (smallest possible inset) and 1 (whole plotting area width or height). The default value for both of these aesthetics is 1/3.

Note

These geoms work only with tibbles as `data`, as they expects a list of graphics objects ("grob") to be mapped to the `label` aesthetic. Aesthetics mappings in the inset plot are independent of those in the base plot.

In the case of `geom_grob()`, `x` and `y` aesthetics determine the position of the whole inset grob, similarly to that of a text label, justification is interpreted as indicating the position of the grob with respect to the $x$ and $y$ coordinates in the data, and `angle` is used to rotate the plot as a whole.

In the case of `geom_grob_npc()`, `npcx` and `npcy` aesthetics determine the position of the whole inset plot, similarly to that of a text label, justification is interpreted as indicating the position of the
grobs with respect to the $x$ and $y$ coordinates in "npc" units, and angle is used to rotate the plot as a whole.

annotate() cannot be used with geom = "grob". Use annotation_custom directly when adding inset plots as annotations.

References

The idea of implementing a geom_custom() for grobs has been discussed as an issue at https://github.com/tidyverse/ggplot2/issues/1399.

See Also

Other Statistics for adding insets to ggplots: geom_plot(), geom_table()

Examples

library(tibble)

df <- tibble(x = 2, y = 15, grob = list(grid::circleGrob(r = 0.2)))
ggplot(data = mtcars, aes(wt, mpg)) +
  geom_point(aes(colour = factor(cyl))) +
  geom_grob(data = df, aes(x, y, label = grob))

---

**geom_label_npc**  
Text with Normalised Parent Coordinates

Description

'geom_text_npc()' adds text directly to the plot. 'geom_label_npc()' draws a rectangle behind the text, making it easier to read. The difference is that x and y mappings are expected to be given in 'npc' graphic units. They are intended to be used for positioning text relative to the physical dimensions of a plot. This can be achieved with 'annotate()' except when facetting is used.

Usage

gem_label_npc(  
  mapping = NULL,  
  data = NULL,  
  stat = "identity",  
  position = "identity",  
  ... ,
  parse = FALSE,  
  nudge_x = 0,  
  nudge_y = 0,  
  label.padding = unit(0.25, "lines"),
  label.r = unit(0.15, "lines"),  
  label.size = 0.25,  
  na.rm = FALSE,  
)
show.legend = FALSE,
    inherit.aes = FALSE
  )

geom_text_npc(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ..., 
  parse = FALSE,
  nudge_x = 0,
  nudge_y = 0,
  check_overlap = FALSE,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = FALSE
  )

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.

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

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

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

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

nudge_x, nudge_y     Horizontal and vertical adjustment to nudge labels by. Useful for offsetting text
                      from points, particularly on discrete scales.

label.padding    Amount of padding around label. Defaults to 0.25 lines.

label.r          Radius of rounded corners. Defaults to 0.15 lines.

label.size       Size of label border, in mm.

na.rm            If FALSE (the default), removes missing values with a warning. If TRUE silently
                 removes missing values.

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`.

check_overlap   If ‘TRUE’, text that overlaps previous text in the same layer will not be plotted.
Details

Note that the “width” and “height” of a text element are 0, so stacking and dodging text will not work by default, and axis limits are not automatically expanded to include all text. Obviously, labels do have height and width, but they are physical units, not data units. The amount of space they occupy on the plot is not constant in data units: when you resize a plot, labels stay the same size, but the size of the axes changes.

`geom_text_npc()` and `geom_label_npc()` add labels for each row in the data, even if coordinates x, y are set to single values in the call to `geom_label_npc()` or `geom_text_npc()`. To add labels at specified points use [annotate()] with `annotate(geom = "text_npc", ...)` or `annotate(geom = "label_npc", ...)`.

`geom_label_npc()`

Currently `geom_label_npc()` does not support the ‘angle’ aesthetic and is considerably slower than `geom_text_npc()`. The ‘fill’ aesthetic controls the background colour of the label.

Alignment

You can modify text alignment with the ‘vjust’ and ‘hjust’ aesthetics. These can either be a number between 0 (right/bottom) and 1 (top/left) or a character ("left", "middle", "right", "bottom", "center", "top"). There are two special alignments: "inward" and "outward". Inward always aligns text towards the center, and outward aligns it away from the center. When using textual positions a shift is added based on grouping, however unused levels are not dropped. In plots with faceting so that not all groups appear in each panel, gaps will appear in between labels. To solve this pass numeric values for the npc coordinates of each label instead of character strings.

Note

This geom is identical to ‘ggplot2’ geom_text() except that it interprets x and y positions in npc units. It translates x and y coordinates from npc units to native data units and calls functions from ‘ggplot2’s GeomText().

See Also

`geom_text`

Examples

df <- data.frame(
  x = c(0, 0, 1, 1, 0.5),
  x.chr = c("left", "left", "right", "right", "center"),
  y = c(0, 1, 0, 1, 0.5),
  y.chr = c("bottom", "top", "bottom", "top", "middle"),
  text = c("bottom-left", "top-left", "bottom-right", "top-right", "center-middle")
)

ggplot(df) +
  geom_text_npc(aes(npcx = x, npcy = y, label = text))

ggplot(df) +
geom_text_npc(aes(npcx = x.chr, npcy = y.chr, label = text))

ggplot(data = mtcars, mapping = aes(wt, mpg)) +
  geom_point() +
  geom_text_npc(data = df, aes(npcx = x, npcy = y, label = text))

ggplot(data = mtcars, mapping = aes(wt, mpg)) +
  geom_point() +
  geom_text_npc(data = df, aes(npcx = x, npcy = y, label = text)) +
  expand_limits(y = 40, x = 6)

ggplot(data = mtcars) +
  geom_point(mapping = aes(wt, mpg)) +
  geom_label_npc(data = df, aes(npcx = x, npcy = y, label = text))

---

geom_plot

Inset plots

Description

**geom_plot** adds ggplot objects as insets to the base ggplot, using syntax similar to that of **geom_label**.

Usage

```r
geom_plot(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...
)

geom_plot_npc(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...
)
```

na.rm = FALSE,
show.legend = FALSE,
inherit.aes = FALSE
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.
- **stat**: The statistical transformation to use on the data for this layer, as a string.
- **position**: Position adjustment, either as a string, or the result of a call to a position adjustment function.
- **na.rm**: If `FALSE` (the default), removes missing values with a warning. If `TRUE` silently removes missing values.
- **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

Note the "width" and "height" like of a text element are 0, so stacking and dodging inset plots will not work by default, and axis limits are not automatically expanded to include all inset plots. Obviously, plots do have height and width, but they are physical units, not data units. The amount of space they occupy on the main plot is not constant in data units of the base plot: when you modify scale limits, inset plots stay the same size relative to the physical size of the base plot.

Inset alignment

You can modify inset plot alignment with the `vjust` and `hjust` aesthetics. These can either be a number between 0 (right/bottom) and 1 (top/left) or a character ("left", "middle", "right", "bottom", "center", "top"). The `angle` aesthetics can be used to rotate the inset plots.

Inset size

You can modify inset plot size with the `vp.width` and `vp.height` aesthetics. These can be a number between 0 (smallest possible inset) and 1 (whole plotting area width or height). The default value for both of these aesthetics is 1/3.

Known problem!

In some cases when explicit coordinates are added to the inner plot, it may be also necessary to add explicitly coordinates to the outer plots.

Note

These geoms work only with tibbles as `data`, as they expects a list of `ggplots` ("gg" objects) to be mapped to the `label` aesthetic. Aesthetics mappings in the inset plot are independent of those in the base plot.
In the case of `geom_plot()`, x and y aesthetics determine the position of the whole inset plot, similarly to that of a text label, justification is interpreted as indicating the position of the plot with respect to the $x$ and $y$ coordinates in the data, and angle is used to rotate the plot as a whole.

In the case of `geom_plot_npc()`, npcx and npcy aesthetics determine the position of the whole inset plot, similarly to that of a text label, justification is interpreted as indicating the position of the plot with respect to the $x$ and $y$ coordinates in "npc" units, and angle is used to rotate the plot as a whole.

`annotate()` cannot be used with `geom = "plot"`. Use `annotation_custom` directly when adding inset plots as annotations.

**References**

The idea of implementing a `geom_custom()` for grobs has been discussed as an issue at [https://github.com/tidyverse/ggplot2/issues/1399](https://github.com/tidyverse/ggplot2/issues/1399).

**See Also**

Other Statistics for adding insets to ggplots: `geom_grob()`, `geom_table()`

**Examples**

```r
# inset plot with enlarged detail from a region of the main plot
library(tibble)

p <- ggplot(data = mtcars, mapping = aes(wt, mpg)) + geom_point()

df <- tibble(x = 0.01, y = 0.01,
  plot = list(p +
    coord_cartesian(xlim = c(3, 4),
    ylim = c(13, 16)) +
    labs(x = NULL, y = NULL) +
    theme_bw(10))))

p +
  expand_limits(x = 0, y = 0) +
  geom_plot_npc(data = df, aes(npcx = x, npcy = y, label = plot))
```

**geom_quadrant_lines**

Reference lines: horizontal plus vertical, and quadrants

**Description**

`geom_vhlines()` adds in a single layer both vertical and horizontal guide lines. Can be thought of as a convenience function that helps with producing consistent vertical and horizontal guide lines. It behaves like `geom_vline()` and `geom_hline()`. `geom_quadrant_lines()` displays the boundaries of four quadrants with an arbitrary origin. The quadrants are specified in the same way as in `stat_quadrant_counts()` and is intended to be used to add guide lines consistent with the counts by quadrant computed by this stat.
geom_quadrant_lines

Usage

geom_quadrant_lines(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  pool.along = "none",
  xintercept = 0,
  yintercept = 0,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = FALSE,
  ...
)

geom_vhlines(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  xintercept = NULL,
  yintercept = NULL,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = FALSE,
  ...
)

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.
stat The statistic object to use display the data
position The position adjustment to use for overlapping points on this layer
pool.along character, one of "none", "x" or "y", indicating which quadrants to pool to calculate counts by pair of quadrants.
xintercept, yintercept numeric vectors the coordinates of the origin of the quadrants.
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? \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 `layer`. This can include aesthetics whose values you want to set, not map. See `layer` for more details.

Details

While `geom_vhlines()` does not provide defaults for the intercepts and accept vectors of length > 1, `geom_quadrant_lines()` sets by default the intercepts to zero producing the natural quadrants and only accepts vectors of length one per panel. That is, `geom_vhlines()` can be used to plot a grid while `geom_quadrant_lines()` plots at most one vertical and one horizontal line. In the case of `geom_quadrant_lines()` the pooling along axes can be specified in the same way as in `stat_quadrant_counts()`.

See Also

`geom_abline`, the topic where `geom_vline()` and `geom_hline()` are described.

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

Examples

```r
# generate artificial data
set.seed(4321)
x <- 1:100
y <- rnorm(length(x), mean = 10)
my.data <- data.frame(x, y)

ggplot(my.data, aes(x, y)) +
  geom_quadrant_lines() +
  geom_point()

ggplot(my.data, aes(x, y)) +
  geom_quadrant_lines(linetype = "dotted") +
  geom_point()

ggplot(my.data, aes(x, y)) +
  geom_quadrant_lines(xintercept = 50, yintercept = 10, colour = "blue") +
  geom_point()

ggplot(my.data, aes(x, y)) +
  geom_quadrant_lines(xintercept = 50, pool.along = "y", colour = "blue") +
  geom_point()

ggplot(my.data, aes(x, y)) +
  geom_vhlines(xintercept = c(25, 50, 75), yintercept = 10, 
              linetype = "dotted", colour = "red") +
  geom_point() +
  theme_bw()
```
**geom_table**

**Inset tables**

**Description**

`geom_table` adds a textual table directly to the ggplot using syntax similar to that of `geom_label` while `geom_table_npc` is similar to `geom_label_npc` in that x and y coordinates are given in npc units.

**Usage**

```r
gemm_table(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ..., 
  parse = FALSE,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = FALSE
)
```

```r
gemm_table_npc(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ..., 
  parse = FALSE,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = FALSE
)
```

**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.
- **stat**: The statistical transformation to use on the data for this layer, as a string.
- **position**: Position adjustment, either as a string, or the result of a call to a position adjustment function.
- **...**: other arguments passed on to `layer`. This can include aesthetics whose values you want to set, not map. See `layer` for more details.
If TRUE, the labels will be parsed into expressions and displayed as described in ?plotmath.

If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.

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.

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.

The "width" and "height" of the table, like for text elements are 0, so stacking and dodging tables will not work by default. In addition, axis limits are not automatically expanded to include the whole tables, but instead only their x and y coordinates. Obviously, tables do have height and width, but they are in physical units, not data units. The amount of space they occupy on a plot is not constant in data units: when you resize a plot, tables stay the same size, but the size of the axes changes.

You can modify table alignment with the vjust and hjust aesthetics. These can either be a number between 0 (right/bottom) and 1 (top/left) or a character ("left", "middle", "right", "bottom", "center", "top").

You can modify inset table size with the size aesthetics, which determines the size of text within the table.

These geoms work only with tibbles as data, as they expects a list of data frames or tibbles ("tb" objects) to be mapped to the label aesthetic. Aesthetics mappings in the inset plot are independent of those in the base plot.

In the case of geom_table(), x and y aesthetics determine the position of the whole inset table, similarly to that of a text label, justification is interpreted as indicating the position of the table with respect to the $x$ and $y$ coordinates in the data, and angle is used to rotate the table as a whole.

In the case of geom_table_npc(), npcx andnpcy aesthetics determine the position of the whole inset table, similarly to that of a text label, justification is interpreted as indicating the position of the table with respect to the $x$ and $y$ coordinates in "npc" units, and angle is used to rotate the table as a whole.

annotate() cannot be used with geom = "table". Use annotation_custom directly when adding inset tables as annotations.
References

This geometry is inspired on answers to two questions in Stackoverflow. In contrast to these earlier examples, the current geom obeys the grammar of graphics, and attempts to be consistent with the behaviour of `ggplot2` geometries.

https://stackoverflow.com/questions/25554548/adding-sub-tables-on-each-panel-of-a-facet-ggplot-in-r?

See Also

function `tableGrob` as it is used to construct the table.

Other Statistics for adding insets to ggplots: `geom_grob()`, `geom_plot()`

Examples

library(dplyr)
library(tibble)

mtcars %>%
group_by(cyl)
summarize(wt = mean(wt), mpg = mean(mpg))
ungroup()
mutate(wt = sprintf("%.2f", wt),
       mpg = sprintf("%.1f", mpg)) -> tb

def <- tibble(x = 0.95, y = 0.95, tb = list(tb))

```r
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table_npc(data = df, aes(npcx = x, npcy = y, label = tb),
                 hjust = 1, vjust = 1)
```

Description

Small arrows on plot margins can supplement a 2d display with annotations. Arrows can be used to highlight specific values along a margin. The geometries `geom_x_margin_arrow()` and `geom_y_margin_arrow()` behave similarly to `geom_vline()` and `geom_hline()` and share their “double personality” as both annotations and geometries.

Usage

```r
geom_x_margin_arrow(
  mapping = NULL,
  data = NULL,
  stat = "identity",
```

```
position = "identity",
..., xintercept,
sides = "b",
arrow.length = 0.03,
na.rm = FALSE,
show.legend = FALSE,
inherit.aes = FALSE
}

geom_y_margin_arrow(
    mapping = NULL,
data = NULL,
    stat = "identity",
    position = "identity",
..., yintercept,
sides = "l",
    arrow.length = 0.03,
    na.rm = FALSE,
    show.legend = FALSE,
inherit.aes = FALSE
)

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.

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

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

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

xintercept, yintercept numeric Parameters that control the position of the marginal points. If these are set, data, mapping and show.legend are overridden.

sides A string that controls which sides of the plot the rugs appear on. It can be set to a string containing any of "trbl", for top, right, bottom, and left.

arrow.length numeric value expressed in npc units for the length of the arrows inwards from the edge of the plotting area.

na.rm If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.

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.
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`.

**See Also**

Other Geometries for marginal annotations in ggplots: `geom_x_margin_grob()`, `geom_x_margin_point()`

**Examples**

```r
p <- ggplot(mtcars, aes(wt, mpg)) + geom_point()
p + geom_x_margin_arrow(xintercept = 3.5)
p + geom_x_margin_arrow(yintercept = c(18, 28, 15))
p + geom_x_margin_arrow(data = data.frame(x = c(2.5, 4.5)),
    mapping = aes(xintercept = x))
p + geom_x_margin_arrow(data = data.frame(x = c(2.5, 4.5)),
    mapping = aes(xintercept = x), sides="tb")
```

**Description**

Marging points can supplement a 2d display with annotations. Marging points can highlight individual cases or values along a margin. The geometries `geom_x_margin_grob()` and `geom_y_margin_grob()` behave similarly `geom_vline()` and `geom_hline()` and share their "double personality" as both annotations and geometries.

**Usage**

```r
geom_x_margin_grob(
    mapping = NULL,
    data = NULL,
    stat = "identity",
    position = "identity",
    ...,
    xintercept,
    sides = "b",
    grob.shift = 0,
    na.rm = FALSE,
    show.legend = FALSE,
    inherit.aes = FALSE
)
```
geom_y_margin_grob(  
  mapping = NULL,  
  data = NULL,  
  stat = "identity",  
  position = "identity",  
  ...,  
  yintercept,  
  sides = "1",  
  grob.shift = 0,  
  na.rm = FALSE,  
  show.legend = FALSE,  
  inherit.aes = FALSE
)

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.

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

position  Position adjustment, either as a string, or the result of a call to a position adjust-

ation function.

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

xintercept, yintercept  numeric Parameters that control the position of the marginal points. If these are set, data, mapping and show.legend are overridden.

sides  A character string of length one that controls on which side of the plot the grob annotations appear on. It can be set to a string containing one of "t", "r", "b" or "l", for top, right, bottom, and left.

grob.shift  numeric value expressed in npc units for the shift of the marginal grob inwards from the edge of the plotting area.

na.rm  If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.

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.

See Also

Other Geometries for marginal annotations in ggplots: `geom_x_margin_arrow()`, `geom_x_margin_point()`
Examples

# We can add icons to the margin of a plot to signal events

---

Description

Marging points can supplement a 2d display with annotations. Marging points can highlight individual cases or values along a margin. The geometries `geom_x_margin_point()` and `geom_y_margin_point()` behave similarly `geom_vline()` and `geom_hline()` and share their "double personality" as both annotations and geometries.

Usage

```r
geom_x_margin_point(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ..., 
  xintercept, 
  sides = "b", 
  point.shift = 0.017, 
  na.rm = FALSE, 
  show.legend = FALSE, 
  inherit.aes = FALSE 
)
```

```r
geom_y_margin_point(
  mapping = NULL, 
  data = NULL, 
  stat = "identity", 
  position = "identity", 
  ..., 
  yintercept, 
  sides = "t", 
  point.shift = 0.017, 
  na.rm = FALSE, 
  show.legend = FALSE, 
  inherit.aes = FALSE 
)
```
**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.
- **stat** The statistical transformation to use on the data for this layer, as a string.
- **position** Position adjustment, either as a string, or the result of a call to a position adjustment function.
- ... other arguments passed on to `layer`. This can include aesthetics whose values you want to set, not map. See `layer` for more details.

**xintercept, yintercept**
numeric Parameters that control the position of the marginal points. If these are set, data, mapping and show.legend are overridden.

- **sides** A string that controls which sides of the plot the rugs appear on. It can be set to a string containing any of “trbl”, for top, right, bottom, and left.
- **point.shift** numeric value expressed in npc units for the shift of the rug points inwards from the edge of the plotting area.
- **na.rm** If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
- **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`.

**See Also**

Other Geometries for marginal annotations in `ggplot2`: `geom_x_margin_point()`, `geom_x_margin_grob()`

**Examples**

```r
p <- ggplot(mtcars, aes(wt, mpg)) +
  geom_point()
p + geom_x_margin_point(xintercept = 3.5)
p + geom_y_margin_point(yintercept = c(18, 28, 15))
p + geom_x_margin_point(data = data.frame(x = c(2.5, 4.5)),
                          mapping = aes(xintercept = x))
p + geom_x_margin_point(data = data.frame(x = c(2.5, 4.5)),
                          mapping = aes(xintercept = x),
                          sides="tb")
```
ggplot() initializes a ggplot object. It can be used to declare the input spectral object for a graphic and to optionally specify the set of plot aesthetics intended to be common throughout all subsequent layers unless specifically overridden.

Usage

## S3 method for class 'ts'

```r
ggplot(
  data,
  mapping = NULL,
  ...,
  time.resolution = "day",
  as.numeric = TRUE,
  environment = parent.frame()
)
```

## S3 method for class 'xts'

```r
ggplot(
  data,
  mapping = NULL,
  ...,
  time.resolution = "day",
  as.numeric = TRUE,
  environment = parent.frame()
)
```

Arguments

data: Default spectrum dataset to use for plot. If not a spectrum, the methods used will be those defined in package ggplot2. See `ggplot`. If not specified, must be supplied in each layer added to the plot.

mapping: Default list of aesthetic mappings to use for plot. If not specified, in the case of spectral objects, a default mapping will be used.

...: Other arguments passed on to methods. Not currently used.

time.resolution: character The time unit to which the returned time values will be rounded.

as.numeric: logical If TRUE convert time to numeric, expressed as fractional calendar years.

environment: If an variable defined in the aesthetic mapping is not found in the data, ggplot will look for it in this environment. It defaults to using the environment in which ggplot() is called.
Details

ggplot() is typically used to construct a plot incrementally, using the + operator to add layers to the existing ggplot object. This is advantageous in that the code is explicit about which layers are added and the order in which they are added. For complex graphics with multiple layers, initialization with ggplot is recommended.

There are three common ways to invoke ggplot:

  • ggplot(ts,aes(x,y,<other aesthetics>))
  • ggplot(ts)

The first method is recommended if all layers use the same data and the same set of aesthetics, although this method can also be used to add a layer using data from another data frame. See the first example below. The second method specifies the default spectrum object to use for the plot, and the units to be used for y in the plot, but no aesthetics are defined up front. This is useful when one data frame is used predominantly as layers are added, but the aesthetics may vary from one layer to another. The third method specifies the default spectrum object to use for the plot, but no aesthetics are defined up front. This is useful when one spectrum is used predominantly as layers are added, but the aesthetics may vary from one layer to another.

Note

Current implementation does not merge default mapping with user supplied mapping. If user supplies a mapping, it is used as is. To add to the default mapping, aes() can be used by itself to compose the ggplot.

Examples

```r
library(ggplot2)
ggplot(lynx) + geom_line()
```

Moved to package 'gginnards'

Description

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

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'.

See Also

gginnards-package, geom_null, stat_debug_group, stat_debug_panel, geom_debug and delete_layers.
outcome2factor

Convert numeric ternary outcomes into a factor

Description

Convert numeric ternary outcomes into a factor

Usage

outcome2factor(x, n.levels = 3L)

threshold2factor(x, n.levels = 3L, threshold = 0)

Arguments

x
a numeric vector of -1, 0, and +1 values, indicating downregulation, uncertain response or upregulation, 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 conveneince 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(), geom_quadrant_lines(), scale_colour_outcome(), scale_shape_outcome(), scale_y_Pvalue(), stat_quadrant_counts(), xy_outcomes2factor() Other scales for omics data: scale_shape_outcome(), scale_x_logFC(), xy_outcomes2factor()

Examples

outcome2factor(c(-1, 1, 0, +1))
outcome2factor(c(-1, 1, 0, +1), n.levels = 2L)

threshold2factor(c(-0.1, -2, 0, +5))
The `quadrant_example.df` dataset contains 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`

**Format**

A data.frame object with 6088 rows and 6 variables

**See Also**

Other Transcriptomics data examples: `volcano_example.df`

**Examples**

```r
names(quadrant_example.df)
head(quadrant_example.df)
```

---

The `scale_colour_outcome` function provides manual scales for colour and fill aesthetics with defaults suitable for the three way outcome from some statistical tests.

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

```r
scale_colour_outcome
```

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

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(), geom_quadrant_lines(), outcome2factor(), scale_shape_outcome(), scale_y_Pvalue(), stat_quadrant_counts(), 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))

library(ggplot2)

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()

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

---

**scale_continuous_npc**  
*Position scales for continuous data (npcx & npcy)*

**Description**

‘scale_npcx_continuous()’ and ‘scale_npcy_continuous()’ are scales for continuous npcx and npcy aesthetics expressed in "npc" units. There are no variants. Obviously limits are always the full range of "npc" units and transformations meaningless. These scales are used by the newly defined aesthetics npcx and npcy.

**Usage**

```r
scale_npcx_continuous()
```

```r
scale_npcy_continuous()
```

**Arguments**

... Other arguments passed on to ‘continuous_scale()’
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()`, `geom_quadrant_lines()`, `outcome2factor()`, `scale_colour_outcome()`, `scale_y_Pvalue()`, `stat_quadrant_counts()`, `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)

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

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

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

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

```r
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.

**Usage**

```r
scale_x_logFC(
  name = "Abundance of x%unit",
  breaks = NULL,
  labels = NULL,
  limits = symmetric_limits,
  oob = scales::squish,
  expand = expand_scale(mult = 0.15, add = 0),
```
scale_x_logFC

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 = expand_scale(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
  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 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

```r
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.
  ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC("A concentration%unit", breaks = log2(c(1/100, 1, 100))) +
  scale_y_logFC("B concentration%unit", breaks = log2(c(1/100, 1, 100)))
```

```r
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()))
```
### scale_y_Pvalue

**Covvenience scale for P-values**

**Description**

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

**Usage**

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

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

```r
  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.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`: Use one of: NULL to use the default scale range, a numeric vector of length two 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()`, `geom_quadrant_lines()`, `outcome2factor()`, `scale_colour_outcome()`, `scale_shape_outcome()`, `stat_quadrant_counts()`, `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))
```

---

**stat_apply_group**

*Apply a function to x or y values*

Description

`stat_apply_group` and `stat_apply_panel` apply functions to data. In most cases one should simply use transformations through scales or summary functions through `stat_summary()`. There are some computations that are not scale transformations but are not usual summaries either, the number of data values does not decrease. It is always possible to precompute quantities like cumulative sums or running medians, and for normalizations it can be convenient to apply such functions on-the-fly to ensure that grouping is consistent between computations and aesthetics. One particularity of these statistics is that they can apply simultaneously different functions to x values and to y values when needed. In contrast `geom_smooth` applies a function that takes both x and y values as arguments.

Usage

```r
stat_apply_group(
  mapping = NULL,
  data = NULL,
  geom = "line",
```
.fun.x = NULL,
.fun.x.args = list(),
.fun.y = NULL,
.fun.y.args = list(),
position = "identity",
na.rm = FALSE,
show.legend = FALSE,
inherit.aes = TRUE,
...)
}

stat_apply_panel(
  mapping = NULL,
  data = NULL,
  geom = "line",
  .fun.x = NULL,
  .fun.x.args = list(),
  .fun.y = NULL,
  .fun.y.args = list(),
  position = "identity",
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  ...
)
)

Arguments

mapping The aesthetic mapping, usually constructed with 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
.fun.x, .fun.y function to be applied or the name of the function to be applied as a character string. One and only one of these parameters should be passed a non-null argument.
.fun.x.args, .fun.y.args additional arguments to be passed to the function as a named list.
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 com-
putation 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.
stat_apply_group

Details

The function(s) to be applied is expected to be vectorized and to return a vector of (almost) the same length. The vector mapped to the x or y aesthetic is passed as the first positional argument to the call. The function must accept as first argument a vector or list that matches the data.

Computed variables

One of x or y or both x and y replaced by the vector returned by the corresponding applied function.

- **x**: x-value as returned by `.fun.x`
- **y**: y-value as returned by `.fun.y`

Note

This stat is at early stages of development and its interface may change at any time.

References


Examples

```r
library(gginnards)
set.seed(123456)
my.df <- data.frame(X = rep(1:20,2),
                    Y = runif(40),
                    category = rep(c("A","B"), each = 20))

# make sure row are ordered for X as we will use functions that rely on this
my.df <- my.df[order(my.df["X"]), ]

ggplot(my.df, aes(x = X, y = Y, colour = category)) +
  stat_apply_group(.fun.y = cumsum)

# Use of geom_debug() to inspect the computed values
ggplot(my.df, aes(x = X, y = Y, colour = category)) +
  stat_apply_group(.fun.y = cumsum, geom = "debug")

ggplot(my.df, aes(x = X, y = Y, colour = category)) +
  stat_apply_group(.fun.y = cummax)

# diff returns a shorter vector by 1 for each group
ggplot(my.df, aes(x = X, y = Y, colour = category)) +
  stat_apply_group(.fun.y = diff, na.rm = TRUE)

ggplot(my.df, aes(x = X, y = Y, colour = category)) +
  geom_point() +
```
stat_apply_group(.fun.y = runmed, .fun.y.args = list(k = 5))

# Rescaling per group
ggplot(my.df, aes(x = X, y = Y, colour = category)) +
   stat_apply_group(.fun.y = function(x) {(x - min(x)) / (max(x) - min(x)))}

# Joint rescaling for whole panel
ggplot(my.df, aes(x = X, y = Y, colour = category)) +
   stat_apply_panel(.fun.y = function(x) {(x - min(x)) / (max(x) - min(x)))}

---

**stat_dens2d_filter**  
*Filter observations by local density*

**Description**

*stat_dens2d_filter* Filters-out/filters-in observations in regions of a plot panel with high density of observations. *stat_dens2d_filter_g* does the filtering by group instead of by panel. This second stat is useful for highlighting observations, while the first one tends to be most useful when the aim is to prevent clashes among text labels.

**Usage**

```r
stat_dens2d_filter(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  keep.fraction = 0.1,
  keep.number = Inf,
  keep.sparse = TRUE,
  na.rm = TRUE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  h = NULL,
  n = NULL,
  ...
)

stat_dens2d_filter_g(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  keep.fraction = 0.1,
  keep.number = Inf,
  keep.sparse = TRUE,
  na.rm = TRUE,
  ...)
```

show.legend = FALSE,
inherit.aes = TRUE,
h = NULL,
n = NULL,

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
keep.fraction numeric [0..1].
keep.number integer number of labels to keep.
keep.sparse logical If false the observations from the densest regions are kept.
na.rm a logical value 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`.
h vector of bandwidths for x and y directions. Defaults to normal reference bandwidth (see bandwidth.nrd). A scalar value will be taken to apply to both directions.
n Number of grid points in each direction. Can be scalar or a length-2 integer vector
...
other arguments passed on to `layer`. This can include aesthetics whose values you want to set, not map. See `layer` for more details.

Computed variables

labels x at centre of range

See Also

kde2d used internally.

Other statistics for selection of observations based on local density: `stat_dens2d_labels()`
Examples

```r
library(ggrepel)
library(gginnards)

random_string <- function(len = 6) {
  paste(sample(letters, len, replace = TRUE), collapse = "")
}

# Make random data.
set.seed(1001)
d <- tibble::tibble(
  x = rnorm(100),
  y = rnorm(100),
  group = rep(c("A", "B"), c(50, 50)),
  lab = replicate(100, { random_string() })
)

ggplot(data = d, aes(x, y)) +
  geom_point() +
  stat_dens2d_filter(color = "red")

# Using geom_debug() we can see that only 10 out of 100 rows in `d` are
# returned. Those highlighted in red in the previous example.

ggplot(data = d, aes(x, y)) +
  geom_point() +
  stat_dens2d_filter(geom = "debug")

# Now, with `keep.fraction = 0.5` we can ensure that only 10 rows are
# returned.

ggplot(data = d, aes(x, y)) +
  geom_point() +
  stat_dens2d_filter(color = "red", keep.fraction = 0.5)

# Now, with `keep.number = 12` we can ensure that only 12 rows are
# returned.

ggplot(data = d, aes(x, y)) +
  geom_point() +
  stat_dens2d_filter(color = "red",
                    keep.fraction = 0.5,
                    keep.number = 12)

ggplot(data = d, aes(x, y, color = group)) +
  geom_point() +
  stat_dens2d_filter(shape = 1, size = 3, keep.fraction = 1/4)

ggplot(data = d, aes(x, y, color = group)) +
  geom_point() +
  stat_dens2d_filter_g(shape = 1, size = 3, keep.fraction = 1/4)

ggplot(data = d, aes(x, y, label = lab, color = group)) +
  geom_point() +
  stat_dens2d_filter(geom = "text")

ggplot(data = d, aes(x, y, label = lab, color = group)) +
  geom_point() +
```

**stat_dens2d_labels**

```r
stat_dens2d_filter(geom = "text_repel")
```

---

**stat_dens2d_labels**  
*Reset labels of observations in high density regions*

**Description**

`stat_low_dens` Sets values mapped to `label` to `""` in regions of a plot panel with high density of observations.

**Usage**

```r
stat_dens2d_labels(
  mapping = NULL,
  data = NULL,
  geom = "text",
  position = "identity",
  keep.fraction = 0.1,
  keep.number = Inf,
  h = NULL,
  n = NULL,
  label.fill = "",
  na.rm = TRUE,
  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
- **keep.fraction** numeric [0..1].
- **keep.number** integer number of labels to keep.
- **h** vector of bandwidths for x and y directions. Defaults to normal reference bandwidth (see `bandwidth.nrd`). A scalar value will be taken to apply to both directions.
- **n** Number of grid points in each direction. Can be scalar or a length-2 integer vector
- **label.fill** character.
stat_dens2d_labels

na.rm a logical value 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_dens2d_labels() is designed to work together with statistics from package ‘ggrepel’. To avoid text labels being plotted over unlabelled points the corresponding rows in data need to be retained but labels replaced with the empty character string, "". stat_dens2d_labels() replaces labels by "" based on the local density of observations.

Computed variables

labels x at centre of range

See Also

kde2d used internally.

Other statistics for selection of observations based on local density: stat_dens2d_filter()

Examples

library(ggrepel)
library(gginnards)

random_string <- function(len = 6) {
  paste(sample(letters, len, replace = TRUE), collapse = "")
}

# Make random data.
set.seed(1001)
d <- tibble::tibble(
  x = rnorm(100),
  y = rnorm(100),
  group = rep(c("A", "B"), c(50, 50)),
  lab = replicate(100, { random_string() })
)

ggplot(data = d, aes(x, y, label = lab)) +
  geom_point() +
  stat_dens2d_labels()

# Using geom_debug() we can see that all 100 rows in \code{d} are
# returned. But only those labelled in the previous example still contain
# the original labels.
ggplot(data = d, aes(x, y, label = lab)) +
  geom_point() +
  stat_dens2d_labels(geom = "debug")

ggplot(data = d, aes(x, y, label = lab, color = group)) +
  geom_point() +
  stat_dens2d_labels()

ggplot(data = d, aes(x, y, label = lab, color = group)) +
  geom_point() +
  stat_dens2d_labels(geom = "text_repel")

ggplot(data = d, aes(x, y, label = lab, color = group)) +
  geom_point() +
  stat_dens2d_labels(geom = "text_repel", label.fill = NA)

---

**stat_fit_augment**  
*Augment data with fitted values and statistics*

**Description**

*stat_fit_augment* fits a model and returns the data augmented with information from the fitted model, using package ‘broom’.

**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.
method.args list of arguments to pass to method.
augment.args list of arguments to pass 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.

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 broom::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.

See Also

`broom`

Other `ggplot2` statistics based on 'broom': `stat_fit_glance()`, `stat_fit_tb()`, `stat_fit_tidy()`

Examples

```r
library(gginnards)
# Regression by panel, using geom_debug() to explore computed variables
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_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

```r
stat_fit_deviations(
  mapping = NULL,
  data = NULL,
  geom = "segment",
  method = "lm",
  formula = 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
- **method**  
Character Currently only "lm" is implemented.
- **formula**  
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 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 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. At the moment it supports only linear models fitted with function lm(). 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, 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.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

For linear models x1 is equal to x2.

See Also

Other statistics for linear model fits: stat_fit_residuals(), stat_poly_eq()
my.data <- data.frame(x, y, group = c("A", "B"), y2 = y * c(0.5, 2))

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

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

# plot, using geom_debug()
ggplot(my.data, aes(x, y)) +
  geom_smooth(method = "lm", formula = my.formula) +
  stat_fit_deviations(formula = my.formula, color = "red", 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 summary "glance" of the model’s statistics, using package 'broom'.

**Usage**

stat_fit_glance(
  mapping = NULL,
  data = NULL,
  geom = "text_npc",
  method = "lm",
  method.args = list(formula = y ~ x),
  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 dataset - only needed if you want to override the plot defaults.
- **geom**: The geometric object to use display the data
- **method**: character.
- **method.args**: list of arguments to pass to `method`.
- **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.

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 apply 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.

Computed variables

The output of `glance()` is returned almost as is in the data object. 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, which vary depending on the model fitting function and model formula we suggest the use of `geom_debug`. An example is shown below.

See Also

- `broom`
- Other `ggplot2` statistics based on 'broom': `stat_fit_augment()`, `stat_fit_tb()`, `stat_fit_tidy()`

Examples

```r
library(gginnards)
# Regression by panel example, using geom_debug.
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\",
                                              stat(r.squared), 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\",
                                              stat(r.squared), stat(p.value)))
```
# Weighted regression example
```r
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("r^2\text{=}%.3f \ P\text{=}%.2g", stat(r.squared), stat(p.value))),
                  parse = TRUE)
```

---

**Description**

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

**Usage**

```r
stat_fit_residuals(
  mapping = NULL,
  data = NULL,
  geom = "point",
  method = "lm",
  formula = NULL,
  resid.type = 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

- **method**
  - character Currently only "lm" is implemented.
formula: a "formula" object. Using aesthetic names instead of original variable names.

resid.type: character passed to `residuals()` as argument for 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.

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 plot residuals as points in a plot. At the moment it supports only linear models fitted with function `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.

\n\n- `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.

See Also

Other statistics for linear model fits: `stat_fit_deviations()`, `stat_poly_eq()`

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, group = c("A", "B"), y2 = y * c(0.5, 2))

# give a name to a formula
my.formula <- y ~ poly(x, 3, raw = TRUE)
```
# plot
ggplot(my.data, aes(x, y)) +
    stat_fit_residuals(formula = my.formula, resid.type = "working")

library(gginnards) # needed for geom_debug()
# print to the console the returned data
ggplot(my.data, aes(x, y)) +
    stat_fit_residuals(formula = my.formula, resid.type = "working",
                      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 package 'broom'. The annotation is added to the plots in tabular form.

**Usage**

```r
stat_fit_tb(
    mapping = NULL,
    data = NULL,
    geom = "table_npc",
    method = "lm",
    method.args = list(formula = y ~ x),
    tb.type = "fit.summary",
    tb.vars = NULL,
    digits = 3,
    label.x = "center",
    label.y = "top",
    label.x.npc = NULL,
    label.y.npc = 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

### method
character.

### method.args
list of arguments to pass to method.

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

### tb.vars
character vector, optionally named, used to select and or rename the columns of the table returned.

### digits
integer indicating the number of significant digits to be used.

### 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

### 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`. An example is shown below.
See Also

`broom` for details on how the tidying of the result of model fits is done. See `geom_table` for details on how the formatting and location of the table can be adjusted.

Other ggplot2 statistics based on `broom`: `stat_fit_augment()`, `stat_fit_glance()`, `stat_fit_tidy()`

Examples

library(gginnards)
# 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
ggplot(my.df, aes(covariate, x)) +
  geom_point() +
  stat_fit_tb() +
  expand_limits(y = 70)

# Linear regression
ggplot(my.df, aes(covariate, x)) +
  geom_point() +
  stat_fit_tb(geom = "debug") +
  expand_limits(y = 70)

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

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

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

# 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
```r
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)
```

---

### 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 package ‘broom’. To add the summary in tabular form use `stat_fit_tb`. When using `stat_fit_tidy()` you will most likely want to change the default mapping for label.

**Usage**

```r
stat_fit_tidy(
  mapping = NULL,
  data = NULL,
  geom = "text_npc",
  method = "lm",
  method.args = list(formula = y ~ x),
  label.x = "left",
  label.y = "top",
  hstep = 0,
  vstep = 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.
**stat_fit_tidy**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geom</td>
<td>The geometric object to use display the data</td>
</tr>
<tr>
<td>method</td>
<td>character.</td>
</tr>
<tr>
<td>method.args</td>
<td>list of arguments to pass to method.</td>
</tr>
<tr>
<td>label.x, label.y</td>
<td>numeric with range 0.1 or character. Coordinates to be used for positioning the output, expressed in &quot;normalized parent coordinates&quot; or character string. If too short they will be recycled.</td>
</tr>
<tr>
<td>hstep, vstep</td>
<td>numeric in npc units, the horizontal and vertical step used between labels for different groups.</td>
</tr>
<tr>
<td>position</td>
<td>The position adjustment to use for overlapping points on this layer</td>
</tr>
<tr>
<td>na.rm</td>
<td>a logical indicating whether NA values should be stripped before the computation proceeds.</td>
</tr>
<tr>
<td>show.legend</td>
<td>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.</td>
</tr>
<tr>
<td>inherit.aes</td>
<td>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.</td>
</tr>
<tr>
<td>...</td>
<td>other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.</td>
</tr>
</tbody>
</table>

**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.

**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.

**Computed variables**

The output of tidy() is returned after reshaping it into a single row. Grouping is respected, and the model fit 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.

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.

**See Also**

broom

Other ggplot2 statistics based on 'broom': stat_fit_augment(), stat_fit_glance(), stat_fit_tb()

**Examples**

```r
library(gginnards)
# Regression by panel, exploring computed variables with geom_debug()
ggplot(mtcars, aes(x = disp, y = mpg)) +
  stat_smooth(method = "lm") +
  geom_point(aes(colour = factor(cyl))) +
  stat_fit_tidy(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_tidy(method = "lm",
                label.x = "right",
                method.args = list(formula = y ~ x),
                mapping = aes(label = sprintf("Slope = %.3g, p-value = %.3g",
                                             stat(x_estimate),
                                             stat(x_p.value))))

# Regression by group 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),
                mapping = aes(label = sprintf("Slope = %.3g, p-value = %.3g",
                                              stat(x_estimate),
                                              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
p-value = %.3g",
    stat(x_estimate),
    stat(x_p.value))))

stat_fmt_tb

Select and slice a tibble nested in data

Description

stat_partial_tb selects columns and/or remanes them and/or slices rows from a tibble nested in data. This stat is designed to be used to pre-process tibble objects mapped to the label aesthetic before adding them to a plot with geom_table.

Usage

stat_fmt_tb(
  mapping = NULL,
  data = NULL,
  geom = "table",
  tb.vars = NULL,
  tb.rows = NULL,
  digits = 3,
  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
tb.vars character vector, optionally named, used to select and or rename the columns of the table returned.
tb.rows integer vector of row indexes of rows to be retained.
digits integer indicating the number of significant digits to be retained in data.
position The position adjustment to use for overlapping points on this layer
stat_peaks

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.

Computed variables

The output of sequentially applying slice with tb.rows as argument and select with tb.vars to a list variable list mapped to label and containing a single tibble per row in data.

Examples

library(ggplot2)
my.df <-
tibble::tibble(
  x = c(1, 2),
  y = c(0, 4),
  group = c("A", "B"),
  tbs = list(a = tibble::tibble(X = 1:6, Y = rep(c("x", "y"), 3)),
             b = tibble::tibble(X = 1:3, Y = "x"))
)

ggplot(my.df, aes(x, y, label = tbs)) +
  stat_fmt_tb() +
  expand_limits(x = c(0, 3), y = c(-2, 6))

ggplot(my.df, aes(x, y, label = tbs)) +
  stat_fmt_tb(tb.vars = c(value = "X", group = "Y"),
              tb.rows = 1:3) +
  expand_limits(x = c(0, 3), y = c(-2, 6))

---

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 x and y numeric values for peaks or valleys and formatted character labels. The formatting is determined by a format string suitable for sprintf().
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,
  ...
)
```

```r
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.

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.

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 POSIXct objects, in which case the x.label.fmt must be suitable for strftime() rather than for 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 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.

See Also
Other peaks and valleys functions: `find_peaks()`

Examples

```r
library(ggplot2)
lynx.df <- data.frame(year = as.numeric(time(lynx)), lynx = as.matrix(lynx))
ggplot(lynx.df, aes(year, lynx)) + geom_line() +
  stat_peaks(colour = "red") +
  stat_valleys(colour = "blue")
ggplot(lynx.df, aes(year, lynx)) + geom_line() +
  stat_peaks(colour = "red") +
  stat_peaks(colour = "red", geom = "rug")
```

### Description

`stat_poly_eq` fits a polynomial and generates several labels including the equation and/or p-value, coefficient of determination ($R^2$), 'AIC' or 'BIC'.

### Usage

```r
stat_poly_eq(
  mapping = NULL,
  data = NULL,
  geom = "text_npc",
  position = "identity",
  ...,
  formula = NULL,
  eq.with.lhs = TRUE,
  eq.x.rhs = NULL,
  coef.digits = 3,
  rr.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,
)```
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
- **formula**: a formula object. Using aesthetic names instead of original variable names.
- **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.
- **coef.digits, rr.digits**: integer Number of significant digits to use in for the vector of fitted coefficients and for $R^2$ labels.
- **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" or "text", or "numeric".
- **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`.

Details

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" 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 only generates labels, 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. `stat_poly_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_poly_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:

- **x.npcx** x position
- **y.npcy** y position
- **coef.ls**, **r.squared**, **adj.r.squared**, **AIC**, **BIC** as numeric values extracted from fit object
- **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
- **AIC.label** AIC for the fitted model.
- **BIC.label** BIC for the fitted model.
- **hjust**, **vjust** Set to "inward" to override the default of the "text" geom.

If output.type is "numeric" the returned tibble contains columns:

- **x.npcx** x position
- **y.npcy** y position
- **coef.ls** list containing the "coefficients" matrix from the summary of the fit object
- **r.squared**, **adj.r.squared**, **AIC**, **BIC** numeric values extracted from fit object
- **hjust**, **vjust** Set to "inward" to override the default of the "text" geom.

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

**Parsing may be required**

if using the computed labels with output.type = "expression", then parse = TRUE is needed, while if using output.type = "LaTeX" parse = FALSE is needed.
Note

For backward compatibility a logical is accepted as argument for `eq.with.lhs`, giving the same output than the current default character value. By default "x" is retained as independent variable as this is the name of the aesthetic. However, it can be substituted by providing a suitable replacement character string through `eq.x.rhs`.

References

Written as an answer to a question at Stackoverflow. [https://stackoverflow.com/questions/7549694/adding-regression-line-equation-and-r2-on-graph](https://stackoverflow.com/questions/7549694/adding-regression-line-equation-and-r2-on-graph)

See Also

This `stat_poly_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 mapping of aesthetic label explicitly supplied in the call.

Other statistics for linear model fits: `stat_fit_deviations()`, `stat_fit_residuals()`

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))

# 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, parse = TRUE)

ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(formula = formula, parse = TRUE,
  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, parse = TRUE,
  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, parse = TRUE)

# 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, rr.digits = 4, parse = TRUE)

# user specified label

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

# user specified label and digits

ggplot(my.data, aes(x, y)) +
geom_point() +
geom_smooth(method = "lm", formula = formula) +
stat_poly_eq(aes(label = paste(stat(eq.label),
stat(adj.rr.label), sep = "-----")),
formula = formula, rr.digits = 3, coef.digits = 2,
parse = TRUE)

# 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, parse = TRUE)

# using numeric values

# Here we use column "Estimate" from the matrix.
# Other available columns are "Std. Error", "t value" and "Pr(>|t|)".
my.format <-
"b[0]~\hat{~}%.3g\hat{,} \"*b[1]~\hat{~}%.3g\hat{,} \"*b[2]~\hat{~}%.3g\hat{,} \"*b[3]~\hat{~}%.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,
stat(coef.ls)[[1]][[1], "Estimate"]],
stat(coef.ls)[[1]][[2], "Estimate"],
stat(coef.ls)[[1]][[3], "Estimate"],
stat(coef.ls)[[1]][[4], "Estimate"]))
# Examples using geom_debug() to show computed values
# This provides a quick way of finding out which variables are available for use in mapping of aesthetics when using other geoms as in the examples above.

library(gginnards)

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

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

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

# 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]]})

description

stat_quadrant_counts() counts the number of observations in each quadrant of a plot panel. By default it adds a text label to the far corner of each quadrant. It can also be used to obtain the total number of observations in each of two pairs of quadrants or in the whole panel. Grouping is ignored, so in every case a single count is computed for each quadrant in a plot panel.

Usage

stat_quadrant_counts(
  mapping = NULL,
  data = NULL,
  geom = "text_npc",
  output.type = "text",
  summary.fun = NULL,
  groups = NA,
  per_group = TRUE,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  number = TRUE,
  number.format = NULL,
  n = NA)
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
quadrants  integer vector indicating which quadrants are of interest, with a `NULL` indicating the whole plot.
pool.along  character, one of "none", "x" or "y", indicating which quadrants to pool to calculate counts by pair of quadrants.
xintercept, yintercept  numeric the coordinates of the origin of the quadrants.
label.x, label.y  numeric Coordinates (in npc units) to be used for absolute positioning of the labels.
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 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 count observations in each of the four quadrants of a plot, and by default add these counts as text labels. Values exactly equal to zero are counted as belonging to the positive quadrant. An argument value of zero, passed to formal parameter `quadrants` is interpreted as a request for the count of all observations in each plot panel.
The default origin of quadrants is at xintercept = 0, yintercept = 0. Also by default, counts are computed for all quadrants within the $x$ and $y$ scale limits, but ignoring any marginal scale expansion. The default positions of the labels is in the farthest corner or edge of each quadrant using npc coordinates. Consequently, when using facets even with free limits for $x$ and $y$ axes, the location of the labels is consistent across panels. This is achieved by use of geom = "text_npc" or geom = "label_npc". To pass the positions in native data units, pass geom = "text" explicitly as argument.

Computed variables

Data frame with one to four rows, one for each quadrant for which counts are counted in data.

- quadrant integer, one of 0:4
- x x value of label position in data units
- y y value of label position in data units
- npcx x value of label position in npc units
- npcy y value of label position in npc units
- count number of observations

As shown in one example below geom_debug can be used to print the computed values returned by any statistic. The output shown includes also values mapped to aesthetics, like label in the example.

See Also

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

Examples

```r
library(gginnards)
# generate artificial data
set.seed(4321)
x <- 1:100
y <- rnorm(length(x), mean = 10)
my.data <- data.frame(x, y)
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quadrant_counts()

# We use geom_debug() to see the computed values
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quadrant_counts(geom = "debug")
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quadrant_counts(aes(label = sprintf("%i observations", stat(count))))
```
**symmetric_limits**  

Expand limits to be symmetric

**Description**

A simple function to expand scale limits to be symmetric around zero. Can be passed as argument to parameter limits of continuous scales from packages `ggplot2` or `scales`.

**Usage**

```r
symmetric_limits(x)
```

**Arguments**

- `x` numeric The automatic limits

**Value**

A numeric vector of length two with the new limits.

**Examples**

```r
symmetric_limits(c(-1, 1.8))
```
**try_data_frame**  
*Convert an R object into a tibble*

**Description**

This functions tries to convert any R object into a data.frame object. If x is already a data.frame, it is returned as is. If it is a list or a vector it is converted by means of `as.data.frame()`. If of any other type, a conversion into an object of class xts is attempted by means of `try.xts()` and if successful the xts object is converted into a data frame with a variable `time` containing times as POSIXct and the remaining data columns with the time series data. In this conversion row names are stripped.

**Usage**

```r
try_data_frame(
  x,
  time.resolution = "month",
  as.numeric = FALSE,
  col.names = NULL
)
```

```r
try_tibble(x, time.resolution = "month", as.numeric = FALSE, col.names = NULL)
```

**Arguments**

- `x` An R object
- `time.resolution` character The time unit to which the returned time values will be rounded.
- `as.numeric` logical If TRUE convert time to numeric, expressed as fractional calendar years.
- `col.names` character vector

**Value**

A tibble::tibble object, derived from data.frame.

**Warning!**

The time zone was set to "UTC" by try.xts() in the test cases I used. Setting TZ to "UTC" can cause some trouble as several frequently used functions have as default the local or system TZ and will apply a conversion before printing or plotting time data, which in addition is affected by summer/winter time transitions. This should be taken into account as even for yearly data when conversion is to POSIXct a day (1st of January) will be set, but then shifted some hours if printed on a TZ different from "UTC". I recommend reading the documentation of package lubridate-package where the irregularities of time data and the difficulties they cause are very well described. In many cases when working with time series with yearly observations it is best to work with numeric values for years.
Note

This function can be used to easily convert time series data into a format that can be easily plotted with package ggplot2. `try_tibble` is another name for `try_data_frame` which tracks the separation and re-naming of `data_frame` into `tibble::tibble` in the imported packages.

Examples

```r
library(xts)
class(lynx)
try_data_frame(lynx)
try_data_frame(lynx, "year")
class(austres)
try_data_frame(austres)
try_data_frame(austres, "quarter")
class(cars)
try_data_frame(cars)
```

---

**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

```r
volcano_example.df
```

Format

A `data.frame` object with 1218 rows and 5 variables

See Also

Other Transcriptomics data examples: `quadrant_example.df`

Examples

```r
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 upregulation, 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(), geom_quadrant_lines(), outcome2factor(), scale_colour_outcome(), scale_shape_outcome(), scale_y_Pvalue(), stat_quadrant_counts()

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

Examples

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

*Topic datasets
  quadrant_example.df, 26
  volcano_example.df, 73

aes, 6, 8, 11, 13, 15, 18, 20, 22, 36, 39, 41, 43, 46, 49, 51, 53, 56, 59, 61, 64, 69
aes__, 6, 8, 11, 13, 15, 18, 20, 22, 36, 41, 43, 46, 49, 51, 53, 56, 59, 61, 64, 69
annotation_custom, 7, 12, 16
append_layers (Moved), 24
borders, 6, 8, 11, 13, 16, 19, 20, 22, 36, 39, 42, 44, 47, 49, 52, 54, 57, 60, 62, 64, 69
bottom_layer (Moved), 24
broom, 45, 50, 55, 58
delete_layers, 24
delete_layers (Moved), 24
extract_layers (Moved), 24
factor, 25, 74
FC_format, 14, 25, 27, 29, 35, 70, 74
find_peaks, 63
geom_abline, 14
geom_debug, 24, 44, 47, 50, 54, 58, 65, 70
geom_debug (Moved), 24
geom_grob, 5, 12, 17
geom_grob_npc (geom_grob), 5
geom_label, 5, 10, 15
geom_label_npc, 7
geom_label_npc, 62
geom_null, 24
geom_null (Moved), 24
geom_plot, 7, 10, 17
geom_plot_npc (geom_plot), 10
geom_quadrant_lines, 12, 25, 27, 29, 35, 70, 74
geom_smooth, 35
geom_table, 7, 12, 15, 55
geom_table_npc (geom_table), 15
gem_text, 9
gem_text_npc (geom_label_npc), 7
gem_text_repel, 62
gem_vlines (geom_quadrant_lines), 12
gem_x_margin_arrow, 17, 20, 22
gem_x_margin_grob, 19, 19, 22
gem_x_margin_point, 19, 20, 21
gem_y_margin_arrow
  (geom_x_margin_arrow), 17
gem_y_margin_grob
  (geom_x_margin_grob), 19
gem_y_margin_point
  (geom_x_margin_point), 21
ggplot, 23, 23
ggpmisc (ggpmisc-package), 3
  ggpmisc-package, 3
ggrepel, 62
kde2d, 39, 42
layer, 6, 8, 11, 14, 15, 18, 20, 22, 36, 39, 42, 44, 47, 49, 52, 54, 57, 60, 62, 64, 69
move_layers (Moved), 24
  Moved, 24
num_layers (Moved), 24
outcome2factor, 14, 25, 27, 29, 32, 35, 70, 74
quadrant_example.df, 26, 73
scale_colour_outcome, 14, 25, 26, 29, 35, 70, 74
scale_continuous, 31, 35
scale_continuous_npc, 28
scale_fill_outcome, 25
scale_fill_outcome
  (scale_colour_outcome), 26
scale_manual, 27, 29
scale_npcx_continuous
  (scale_continuous_npc), 28
scale_npcy_continuous
  (scale_continuous_npc), 28
scale_shape_outcome, 14, 25, 27, 29, 32, 35,
  70, 74
scale_x_FDR (scale_y_Pvalue), 33
scale_x_logFC, 25, 29, 30, 74
scale_x_Pvalue (scale_y_Pvalue), 33
scale_y_FDR (scale_y_Pvalue), 33
scale_y_logFC (scale_x_logFC), 30
scale_y_Pvalue, 14, 25, 27, 29, 33, 70, 74
select, 60
shift_layers (Moved), 24
slice, 60
sprintf, 62
stat_apply_group, 35
stat_apply_panel (stat_apply_group), 35
stat_debug_group, 24
stat_debug_group (Moved), 24
stat_debug_panel, 24
stat_debug_panel (Moved), 24
stat_dens2d_filter, 38, 42
stat_dens2d_filter_g
  (stat_dens2d_filter), 38
stat_dens2d_labels, 39, 41
stat_fit_augment, 43, 49, 50, 55, 57, 58
stat_fit_deviations, 46, 52, 66
stat_fit_glance, 44, 45, 48, 55, 57, 58, 66
stat_fit_residuals, 47, 51, 66
stat_fit_tb, 45, 50, 53, 56, 58
stat_fit_tidy, 44, 45, 49, 50, 55, 56, 66
stat_fmt_tb, 59
stat_peaks, 60
stat_poly_eq, 44, 47, 49, 52, 57, 63
stat_quadrant_counts, 14, 25, 27, 29, 35,
  68, 74
stat_valleys (stat_peaks), 60
strftime, 62
strptime, 62
symmetric_limits, 71

tableGrob, 17
threshold2factor (outcome2factor), 25
top_layer (Moved), 24
try_data_frame, 72
try_tibble (try_data_frame), 72
volcano_example.df, 26, 73
which_layers (Moved), 24
xy_outcomes2factor, 14, 25, 27, 29, 32, 35,
  70, 74
xy_thresholds2factor
  (xy_outcomes2factor), 74

which_layers (Moved), 24
xy_outcomes2factor, 14, 25, 27, 29, 32, 35,
  70, 74
xy_thresholds2factor
  (xy_outcomes2factor), 74