Package 'ggpmisc'

April 4, 2021

Type Package

Title Miscellaneous Extensions to 'ggplot2'

Version 0.3.9

Date 2021-04-03

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 (>= 3.6.0), ggplot2 (>= 3.3.2)

Imports grid, rlang (>= 0.4.7), magrittr (>= 1.5), generics (>= 0.1.0), glue (>= 1.4.2), gridExtra (>= 2.3), scales (>= 1.1.1), MASS (>= 7.3-51.6), polynom (>= 1.4-0), splus2R (>= 1.2-2), tibble (>= 3.0.3), plyr (>= 1.8.6), dplyr (>= 1.0.2), xts (>= 0.12-0), zoo (>= 1.8-8), lubridate (>= 1.7.9), stringr (>= 1.4.0)

Suggests knitr (>= 1.29), rmarkdown (>= 2.3), broom (>= 0.7.3), broom.mixed (>= 0.2.6), nlme (>= 3.1-148), gginnards(>= 0.0.3), ggrepel (>= 0.9.1), magick (>= 2.6.0), quantreg (>= 5.8.3)

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

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

Encoding UTF-8

RoxygenNote 7.1.1

VignetteBuilder knitr

NeedsCompilation no

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

Repository CRAN

Date/Publication 2021-04-04 04:40:02 UTC

R topics documented:

`ggpmisc-package` *ggpmisc: Miscellaneous Extensions to 'ggplot2'*

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.

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_layesr()`, `shift_layers()`, `top_layer()` and `which_layers()` have been moved from package 'ggpmisc' into their own seperate 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/tidyverse/ggplot2

See Also

Useful links:

- https://docs.r4photobiology.info/ggpmisc/
- https://github.com/aphalo/ggpmisc
- Report bugs at https://github.com/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 = stat(eq.label)),
               formula = formula,
               parse = TRUE) +
  labs(x = expression("Speed, "*x~("mph")),
       y = expression("Stopping distance, "*y~("ft")))

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

---

annotate *Annotations supporting NPC*

Description

A revised version of annotate() from package 'ggplot2' adding support for npcx and npcy position aesthetics, allowing use of the geometries defined in the current package such as geom_text_npc(). It also has a parameter label that directly accepts data frames, ggplots and grobs as arguments in addition to objects of atomic classes like character. When package 'ggpmisc' is loaded this definition of annotate() overrides that in package 'ggplot2'.

Usage

```
annotate(
  geom,
```

```
  x = NULL,
  y = NULL,
  xmin = NULL,
  xmax = NULL,
  ymin = NULL,
  ymax = NULL,
  xend = NULL,
  yend = NULL,
  npcx = NULL,
  npcy = NULL,
  label = NULL,
  ...,
  na.rm = FALSE
)
```

Arguments

`geom`	character Name of geom to use for annotation.
`x, y, xmin, ymin, xmax, ymax, xend, yend, npcx, npcy`	numeric Positioning aesthetics - you must specify at least one of these.
`label`	character, data.frame, ggplot or grob.
`...`	Other named arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat.
`na.rm`	logical If `FALSE`, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

Details

Note that all position aesthetics are scaled (i.e., they will expand the limits of the plot so they are visible), but all other aesthetics are set. This means that layers created with this function will never affect the legend.

Note

To use the original definition of `annotate()` after loading package 'ggpmisc', use `ggplot2::annotate()`.

Examples

```
p <- ggplot(mtcars, aes(x = wt, y = mpg)) + geom_point()

# Works as ggplot2::annotate()
p + annotate("text", x = 5, y = 32, label = "Some text")
p + annotate("label", x = c(2, 5), y = c(15, 32),
             label = c("A", "B"))
p + annotate("table", x = 5, y = 30,
             label = data.frame(A = 1:2, B = letters[1:2]))
p + annotate("plot", x = 5.5, y = 34,
```

```
            label = p + theme_bw(9))
p + annotate("rect", xmin = 3, xmax = 4.2, ymin = 12, ymax = 21, alpha = .2)
p + annotate("segment", x = 2.5, xend = 4, y = 15, yend = 25, colour = "blue")
p + annotate("pointrange", x = 3.5, y = 20, ymin = 12, ymax = 28,
  colour = "red", size = 1.5)

# But ggpmisc::annotate() also works with npcx and npcy pseudo-aesthetics
p + annotate("label_npc", npcx = c(0.1, 0.9), npcy = c(0.1, 0.9),
            label = c("A", "B"))
p + annotate("label_npc", npcx = 0.9, npcy = c(0.1, 0.9),
            label = c("A", "B"))

p + annotate("text_npc", npcx = 0.9, npcy = 0.9, label = "Some text")
p + annotate("text_npc", npcx = "right", npcy = "top", label = "Some text")

p + annotate("table_npc", npcx = 0.9, npcy = 0.9,
            label = data.frame(A = 1:2, B = letters[1:2]))

p + annotate("plot_npc", npcx = 1, npcy = 1,
            label = p + theme_bw(9))
p + annotate("plot_npc", npcx = c(0, 1), npcy = c(0, 1),
            label = list(p + theme_bw(9), p + theme_grey(9)),
            vp.width = 0.3, vp.height = 0.4)
```

---

geom_grob *Inset graphical objects*

Description

geom_grob and geom_grob_npc add a Grob as inset to the ggplot using syntax similar to that of geom_label.In most respects they behave as any other ggplot geometry: a layer con contain multiple tables and faceting works as usual.

Usage

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

geom_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

The "width" and "height" of an inset as for 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, insets 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.

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 posisble inset) and 1 (whole plotting area width or height). The default value for 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 grob 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 geometries adding layers with insets: `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 faceting is used.

Usage

```
geom_label_npc(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  parse = FALSE,
  nudge_x = 0,
  nudge_y = 0,
  label.padding = grid::unit(0.25, "lines"),
  label.r = grid::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 data set - 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.
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_linked_text *Linked Text*

Description

Text geoms are useful for labelling plots. ‘geom_linked_text()‘ adds text to the plot and for nudged positions links the original location to the nudged text with a segment.

Usage

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

```
  parse = FALSE,
  nudge_x = 0,
  nudge_y = 0,
  arrow = NULL,
  check_overlap = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes or aes_. If specified and inherit.aes = TRUE (the default), is combined with the default mapping at the top level of the plot. You only need to supply mapping if there isn't a mapping defined for the plot.
data	A data frame. If specified, overrides the default data frame defined at the top level of the plot.
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. There are three types of arguments you can use here: • Aesthetics: to set an aesthetic to a fixed value, like colour = "red" or size = 3. • Other arguments to the layer, for example you override the default stat associated with the layer. • Other arguments passed on to the stat.
parse	If TRUE, the labels will be parsed into expressions and displayed as described in ?plotmath.
nudge_x, nudge_y	Horizontal and vertical adjustments to nudge the starting position of each text label. The units for nudge_x and nudge_y are the same as for the data units on the x-axis and y-axis.
arrow	specification for arrow heads, as created by arrow
check_overlap	If TRUE, text that overlaps previous text in the same layer will not be plotted. check_overlap takes place at draw time and in the order of the data, thus its action depends of the size at which the plot is drawn.
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 that when you resize a plot, text labels stay the same size, even though the size of the plot area changes. This happens because the "width" and "height" of a text element are 0. Obviously, text labels do have height and width, but they are physical units, not data units. For the same reason, stacking and dodging text will not work by default, and axis limits are not automatically expanded to include all text.

By default this geom uses ‘position_nudge_center()‘ which is backwards compatible with ‘position_nudge()‘ from ’ggplot2’ but provides additional control on the direction of the nudging. In contrast to ‘position_nudge()‘, ‘position_nudge_center()‘ and ‘position_nudge_line()‘ preserve the original coordinates.

Under development

This is a very simple and preliminary version of a geom. I plan to add features like padding around text and points. I aim to make use of the new features of ’grid’ in R >= 4.1.0 to keep the implementation as fast and simple as possible. Currently this geom does all drawing using at most two vectorized calls to ’grid’ functions. As a temporary replacement of padding around text one can use ’slightly out-of-range’ numeric values for justification as shown in the examples.

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.

Examples

```
my.cars <- mtcars[c(TRUE, FALSE, FALSE, FALSE), ]
my.cars$name <- rownames(my.cars)
p <- ggplot(my.cars, aes(wt, mpg, label = name))

# default behavior is as for geon_text()
p + geom_linked_text()
# Avoid overlaps
p + geom_linked_text(check_overlap = TRUE)
# Change size of the label
p + geom_linked_text(size = 2.5)

# Use nudging
p +
  geom_point() +
  geom_linked_text(hjust = -0.04, nudge_x = 0.12) +
  expand_limits(x = 6.2)
p +
  geom_point() +
  geom_linked_text(hjust = -0.04, nudge_x = 0.12,
  arrow = arrow(length = grid::unit(1.5, "mm"))) +
  expand_limits(x = 6.2)
```

```
p +
  geom_point() +
  geom_linked_text(vjust = -0.5, nudge_y = 0.5)
p +
  geom_point() +
  geom_linked_text(hjust = -0.02, nudge_x = 0.1,
                   vjust = -0.2, nudge_y = 0.5)
p +
  geom_point() +
  geom_linked_text(angle = 90,
                   hjust = -0.04, nudge_y = 1,
                   arrow = arrow(length = grid::unit(1.5, "mm"))) +
  expand_limits(y = 40)

# Add aesthetic mappings
p +
  geom_point() +
  geom_linked_text(aes(colour = factor(cyl)),
                   angle = 90,
                   hjust = -0.04, nudge_y = 1,
                   arrow = arrow(length = grid::unit(1.5, "mm"))) +
  scale_colour_discrete(l = 40) +
  expand_limits(y = 40)

p + geom_linked_text(aes(size = wt)) +
    expand_limits(x = c(2, 6))
# Scale height of text, rather than sqrt(height)
p +
  geom_linked_text(aes(size = wt)) +
  scale_radius(range = c(3,6)) +
    expand_limits(x = c(2, 6))

# You can display expressions by setting parse = TRUE.  The
# details of the display are described in ?plotmath, but note that
# geom_linked_text uses strings, not expressions.
p +
  geom_linked_text(
    aes(label = paste(wt, "^(", cyl, ")", sep = "")),
    parse = TRUE
  )

# Add a text annotation
p +
  geom_linked_text() +
  annotate(
    "linked_text", label = "plot mpg vs. wt",
    x = 2, y = 15, size = 3, colour = "red"
  )  +
 expand_limits(x = c(1.5, 6))

# Justification -----------------------------------------------------------
df <- data.frame(
  x = c(1, 1, 2, 2, 1.5),
```

```
  y = c(1, 2, 1, 2, 1.5),
  text = c("bottom-left", "bottom-right", "top-left", "top-right", "center")
)
ggplot(df, aes(x, y)) +
  geom_linked_text(aes(label = text))
ggplot(df, aes(x, y)) +
  geom_linked_text(aes(label = text), vjust = "inward", hjust = "inward")
```

---

geom_plot *Inset plots*

Description

geom_plot and geom_plot_npc add ggplot objects as insets to the base ggplot, using syntax similar to that of geom_label. In most respects they behave as any other ggplot geometry: a layer con contain multiple tables and faceting works as usual.

Usage

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

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 data set - 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

The "width" and "height" of an inset as for 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, insets 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 and aspect

You can modify the size of the inset plot with the vp.width and vp.height aesthetics. Arguments can be a number between 0 (smallest possible inset) and 1 (whole plotting area width or height). The default value for for both of these aesthetics is 1/3. If the coordinates are "free" the plot stretches to fill the viewport. However, if the coordinates of the inset are "fixed" and the aspect ratio of the viewport is different to that of the inset, the viewport will be surrounded on either x or y margins by invisible space, which may look as if the position of the inset is wrong.

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.

See Also

Other geometries adding layers with insets: `geom_grob()`, `geom_table()`

Examples

```
# 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))

p +
  expand_limits(x = 0, y = 0) +
  geom_plot_npc(data = df,
                vp.width = 1/2, vp.height = 1/4,
                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.

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 aes or aes_. Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific data set - only needed if you want to override the plot defaults.
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? `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

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

```
# 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. In most respects they behave as any other ggplot geometry: a layer con contain multiple tables and faceting works as usual.

Usage

```
geom_table(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  table.theme = NULL,
  table.rownames = FALSE,
  table.colnames = TRUE,
  table.hjust = 0.5,
  parse = FALSE,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = FALSE
)

geom_table_npc(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  table.theme = NULL,
  table.rownames = FALSE,
  table.colnames = TRUE,
  table.hjust = 0.5,
  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 data set - 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
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
table.theme	NULL, list or function A gridExtra ttheme defintion, or a constructor for a ttheme or NULL for default.
table.rownames, table.colnames	logical flag to enable or disable printing of row names and column names.
table.hjust	numeric Horizontal justification for the core and column headings of the table.
parse	If TRUE, the labels will be parsed into expressions and displayed as described in ?plotmath.
na.rm	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

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 and npcy 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.

The "width" and "height" of an inset as for a text element are 0, so stacking and dodging inset tables will not work by default, and axis limits are not automatically expanded to include all inset tables. Obviously, insets 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.

If the argument passed to `table.theme` is a constructor function (passing its name without parenthesis), the values mapped to `size`, `colour`, `fill`, `alpha`, and `family` aesthetics will the passed to this theme constructor for each individual table. In contrast, if a ready constructed ttheme as a list object is passed as argument (e.g., by calling the constructor, using contructor name followed by parenthesis), it will be used as is, i.e., mappings to aesthetics such as `colour` are ignored if present.

By default the constructor `ttheme_gtdefault` is used and `colour` and `fill`, are mapped to `NA`. Mapping these aesthetics to `NA` triggers the use of the default base_colour of the `ttheme`.

As the table is built with function `gridExtra::gtable()`, for formatting details, please, consult `tableGrob`.

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 table size with the `size` aesthetics, which determines the size of text within the table.

Note

As all geometries, `geom_table()` and `geom_table_npc()` add a layer to a plot, and behave as expected in the grammar of graphics. In general ggplot themes do not affect how layers are rendered, and this is also the case for `geom_table()`. As described abovem, the formatting of the inset table is done according to the the argument passed to parameter `table.theme`.

Complex tables with annotations or different colouring of rows or cells can be constructed with functions in package 'gridExtra' or in any other way as long as they can be saved as grid graphical objects and then added to a ggplot as a new layer with `geom_grob`.

In 'ggpmisc' (>= 0.3.6) `annotate()` **can be used with** `geom = "table"`.

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/12318120/adding-table-within-the-plotting-region-of-a-ggplot-in-r 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 geometries adding layers with insets: `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

df <- tibble(x = 5.45, y = 34, tb = list(tb))

# using defaults
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df, aes(x = x, y = y, label = tb))

ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df, aes(x = x, y = y, label = tb),
             table.rownames = TRUE, table.theme = ttheme_gtstripes)

# settings aesthetics to constants
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df, aes(x = x, y = y, label = tb),
             color = "red", fill = "#FFCCCC", family = "serif", size = 5,
             angle = 90, vjust = 0)

# passing a theme constructor as argument
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df, aes(x = x, y = y, label = tb),
             table.theme = ttheme_gtminimal) +
  theme_classic()

df2 <- tibble(x = 5.45, y = c(34, 29, 24), cyl = c(4, 6, 8),
              tb = list(tb[1, 1:3], tb[2, 1:3], tb[3, 1:3]))

# mapped aesthetics
ggplot(data = mtcars, mapping = aes(wt, mpg, color = factor(cyl))) +
  geom_point() +
  geom_table(data = df2,
             inherit.aes = TRUE,
             mapping = aes(x = x, y = y, label = tb))

# Using native plot coordinates instead of data coordinates
dfnpc <- tibble(x = 0.95, y = 0.95, tb = list(tb))

ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
```

```
geom_table_npc(data = dfnpc, aes(npcx = x, npcy = y, label = tb))
```

`geom_x_margin_arrow` *Reference arrows on the margins*

Description

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

Usage

```
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 arows 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_grob(), geom_x_margin_point()

Examples

```
p <- ggplot(mtcars, aes(wt, mpg)) +
  geom_point()
p
p + geom_x_margin_arrow(xintercept = 3.5)
p + geom_y_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")
```

---

geom_x_margin_grob *Add Grobs on the margins*

Description

Marging points can supplement a 2d display with annotations. Marging points can highligth 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

```
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 = "l",
  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 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 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
```

`geom_x_margin_point` *Reference points on the margins*

Description

Marging points can supplement a 2d display with annotations. Marging points can highligth 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

```
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
)

geom_y_margin_point(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  yintercept,
  sides = "l",
  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 ggplots: geom_x_margin_arrow(), geom_x_margin_grob()

Examples

```
p <- ggplot(mtcars, aes(wt, mpg)) +
  geom_point()
p
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 *Create a new ggplot plot from time series data*

Description

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

## S3 method for class 'xts'
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 suppled 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

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

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

Details

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

Note

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

See Also

Other Functions for quadrant and volcano plots: `FC_format()`, `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))
threshold2factor(c(-0.1, -2, 0, +5), n.levels = 2L)
threshold2factor(c(-0.1, -2, 0, +5), threshold = c(-1, 1))
```

---

position_nudge_center *Nudge labels away from a central point*

Description

'position_nudge_center()' is generally useful for adjusting the position of labels or text, both on a discrete or continuous scale. In contrast to [ggplot2::position_nudge], 'position_nudge_center()' returns in 'data' both the original coordinates and the nudged coordinates.

Usage

```
position_nudge_center(
  x = 0,
  y = 0,
  center_x = NULL,
  center_y = NULL,
  direction = NULL,
  obey_grouping = NULL
)

position_nudge_centre(
  x = 0,
  y = 0,
  center_x = NULL,
  center_y = NULL,
```

```
  direction = NULL,
  obey_grouping = NULL
)

position_nudge_keep(x = 0, y = 0)
```

Arguments

x, y	Amount of vertical and horizontal distance to move. A numeric vector of length 1, or of the same length as rows there are in ‘data‘,
center_x, center_y	The coordinates of the virtual origin out from which nudging radiates or splits in opposite directions. A numeric vector of length 1 or of the same length as rows there are in ‘data‘, or a function returning either of these vectors computed from the variables in data mapped to ‘x‘ or ‘y‘, respectively.
direction	One of "none", "radial", or "split". A value of "none" replicates the behavior of [ggplot2::position_nudge]. Which of these three values is the default depends on the values passed to the other parameters.
obey_grouping	A logical flag indicating whether to obey or not groupings of the observations. By default, grouping is obeyed when both of the variables mapped to _x_ and _y_ are continuous numeric and ignored otherwise.

Details

This position function is backwards compatible with [ggplot2::position_nudge] but extends it by adding support for nudging that varies across the plotting region, either in opposite directions or radially from a virtual _center point_.

The wrapper ‘position_nudge_keep()‘ with exactly the same signature and behaviour as [ggplot2::position_nudge] provides an easier to remember name when the desire is only to have access to both the original and nudged coordinates.

Positive values as arguments to ‘x‘ and ‘y‘ are added to the original position along either axis. If no arguments are passed to ‘center_x‘, ‘center_y‘ or ‘direction‘, the nudging is applied as is, as is the case if ‘direction = "none"‘. If non-‘NULL‘ arguments are passed to both ‘center_x‘ and ‘center_y‘, ‘direction = "radial"‘ is assumed. In this case, if ‘x‘ and/or ‘y‘ positive nudging is applied radially outwards from the center, while if negative, inwards towards the center. When a non-‘NULL‘ argument is passed only to one of ‘center_x‘ or ‘center_y‘, ‘direction = "split"‘ is assumed. In this case when the initial location of the point is to the left of ‘center_x‘, ‘-x‘ is used instead of ‘x‘ for nudging, and when the initial location of the point is to the below of ‘center_y‘, ‘-y‘ is used instead of ‘y‘ for nudging. If non-‘NULL‘ arguments are passed to both ‘center_x‘ and ‘center_y‘, and ‘direction‘ is passed ‘"split"‘ as argument, then the split as described above is applied to both _x_ and _y_ coordinates.

Note

Some situations are handled as special cases. When ‘direction = "split"‘ or ‘direction = "radial"‘, observations at exactly the _center_ are nudged using ‘x‘ and ‘y‘ unchanged. When ‘direction = "split"‘, and both ‘center_x‘ and ‘center_y‘ have been supplied, segments are drawn at eight different possible angles. When segments are exactly horizontal or vertical they would be shorter

than when drawn at the other four angles, in which case ‘x‘ or ‘y‘ are extended to ensure these segments are of the same lengths as those at other angles.

This position is most useful when labeling points forming a cloud or along vertical or horizontal lines or "divides".

See Also

[ggplot2::position_nudge()], [ggrepel::position_nudge_repel()].

Other position adjustments: `position_nudge_line()`, `position_nudge_to()`

Examples

```
df <- data.frame(
  x = c(1,3,2,5,4,2.5),
  y = c("abc","cd","d","c","bcd","a")
)

# Plain nudging, same as with ggplot2::position_nudge()

ggplot(df, aes(x, y, label = y)) +
  geom_point() +
  geom_text(hjust = 0, vjust = 0,
            position = position_nudge(x = 0.05, y = 0.07)
  )

ggplot(df, aes(x, y, label = y)) +
  geom_point() +
  geom_text(hjust = 0, vjust = 0,
            position = position_nudge_center(x = 0.05, y = 0.07)
  )

# "split" nudging

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text(aes(label = y),
            hjust = "outward", vjust = "outward",
            position = position_nudge_center(x = 0.05,
                                             y = 0.07,
                                             direction = "split"))

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text(aes(label = y),
            hjust = "outward",
            position = position_nudge_center(x = 0.08,
                                             direction = "split"))

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text(aes(label = y),
            vjust = "outward",
```

```
            position = position_nudge_center(y = 0.1,
                                             direction = "split"))

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text(aes(label = y),
            vjust = "outward", hjust = "outward",
            position = position_nudge_center(x = 0.06,
                                             y = 0.08,
                                             center_y = 2,
                                             center_x = 1.5,
                                             direction = "split"))

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text(aes(label = y),
            vjust = "outward", hjust = "outward",
            position = position_nudge_center(x = 0.06,
                                             y = 0.08,
                                             center_y = 2))

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text(aes(label = y),
            vjust = "outward", hjust = "outward",
            position = position_nudge_center(x = 0.1,
                                             center_x = 2.5))

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text(aes(label = y),
            vjust = "outward", hjust = "outward",
            position = position_nudge_center(x = 0.06,
                                             y = 0.08,
                                             center_x = median,
                                             center_y = median,
                                             direction = "split"))

# "Radial" nudging

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text(aes(label = y),
            vjust = "outward", hjust = "outward",
            position = position_nudge_center(x = 0.1,
                                             y = 0.2,
                                             direction = "radial"))

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text(aes(label = y),
            vjust = "inward", hjust = "inward",
            position = position_nudge_center(x = -0.1,
```

```
                                          y = -0.1,
                                          direction = "radial"))

df <- data.frame(
  x = -10:10,
  z = (-10:10)^2,
  y = letters[1:21],
  group = rep(c("a", "b"), rep(c(11, 10)))
)

ggplot(df, aes(x, z)) +
  geom_point() +
  geom_line() +
  geom_text(aes(label = y),
            vjust = "inward", hjust = "inward",
            position = position_nudge_center(x = -0.9,
                                          y = -2.7,
                                          center_x = mean,
                                          center_y = max))

ggplot(df, aes(x, z)) +
  geom_point() +
  geom_line() +
  geom_text(aes(label = y),
            vjust = "outward", hjust = "outward",
            position = position_nudge_center(x = 0.9,
                                          y = 2.7,
                                          center_x = mean,
                                          center_y = max))

above_max <- function(x) {1.2 * max(x)}
ggplot(df, aes(x, z)) +
  geom_point() +
  geom_line() +
  geom_text(aes(label = y),
            vjust = "inward", hjust = "inward",
            position = position_nudge_center(x = -1.2,
                                          y = -3,
                                          center_x = mean,
                                          center_y = above_max))

ggplot(df, aes(x, z, color = group)) +
  geom_point() +
  geom_line(color = "black", linetype = "dotted") +
  geom_text(aes(label = y),
            vjust = "inward", hjust = "inward",
            position = position_nudge_center(x = -0.9,
                                          y = -2.7,
                                          center_x = mean,
                                          center_y = max))

ggplot(df, aes(x, z, color = group)) +
  geom_point() +
```

```
  geom_line(color = "black", linetype = "dotted") +
  geom_text(aes(label = y),
            vjust = "inward", hjust = "inward",
            position = position_nudge_center(x = -0.9,
                                             y = -2.7,
                                             center_x = mean,
                                             center_y = max,
                                             obey_grouping = FALSE))
```

`position_nudge_line` *Nudge labels away from a line*

Description

'position_nudge_line' is generally useful for adjusting the starting position of labels or text to be repelled while preserving the original position as the start of the segments. The difference compared to [position_nudge_center()] is that the nudging is away from from a line or curve fitted to the data points or supplied as coefficients. While [position_nudge_center()] is most useful for "round-shaped", vertically- or horizontally elongated clouds of points, [position_nudge_line()] is most suitable when observations follow a linear or curvilinear relationship between _x_ and _y_ values. In contrast to [ggplot2::position_nudge], 'position_nudge_line()' returns in 'data' both the original coordinates and the nudged coordinates.

Usage

```
position_nudge_line(
  x = NA_real_,
  y = NA_real_,
  xy_relative = c(0.03, 0.03),
  abline = NULL,
  method = NULL,
  formula = y ~ x,
  direction = NULL,
  line_nudge = 1
)
```

Arguments

`x, y`	Amount of vertical and horizontal distance to move. A numeric vector of length 1, or of the same length as rows there are in 'data'.
`xy_relative`	Nudge relative to _x_ and _y_ data expanse, ignored unless 'x' and 'y' are both 'NA's.
`abline`	a vector of length two giving the intercept and slope.
`method`	One of '"spline"', '"lm"' or '"auto"'.
`formula`	A model formula for [lm()] when 'method = "lm"'. Ignored otherwise.

direction	One of "none", or "split".
line_nudge	A positive multiplier >= 1, increasing nudging away from the curve or line compared to nudging from points.

Details

The default ammount of nudging is 3 _x_ and _y_ axes, which in most cases is good. In most cases it is best to apply nudging along a direction perpendicular to the line or curve, if this is the aim, passing an argument to only one of ‘x‘, ‘y‘ or ‘xy_relative‘ will be enough. When ‘direction = "split"‘ nudging is away from an implicit line or curve on either side with positive nudging. The line of curve can be smooth spline or linear regression fitted on-the-fly to the data points, or a straight line defined by its coefficients passed to ‘abline‘. The fitting is well defined only if the observations fall roughly on a curve or straight line that is monotonic in ‘y‘. By means of ‘line_nudge‘ one can increment nudging away from the line or curve compared to away from the points, which is useful for example to keep labels outside of a confidence band. Direction defaults to ‘"split"‘ when ‘line_nudge > 1‘, and otherwise to ‘"none"‘.

Note

For ‘method = "lm"‘ only model formulas corresponding to polynomials with no missing terms are supported. If using [poly()], ‘raw = TRUE‘ is required.

In practice, ‘x‘ and ‘y‘ should have the same sign for nudging to work correctly.

This position is most useful when labeling points conforming a cloud along an arbitrary curve or line.

See Also

[ggplot::position_nudge()], [ggrepel::position_nudge_repel()].

Other position adjustments: `position_nudge_center()`, `position_nudge_to()`

Examples

```
set.seed(16532)
df <- data.frame(
  x = -10:10,
  y = (-10:10)^2,
  yy = (-10:10)^2 + rnorm(21, 0, 4),
  yyy = (-10:10) + rnorm(21, 0, 4),
  l = letters[1:21]
)

# Setting the nudging distance

ggplot(df, aes(x, y, label = l)) +
  geom_line(linetype = "dotted") +
  geom_point() +
  geom_text(position = position_nudge_line())

ggplot(df, aes(x, y, label = l)) +
```

```
  geom_line(linetype = "dotted") +
  geom_point() +
  geom_text(position = position_nudge_line(xy_relative = -0.03))

ggplot(df, aes(x, y, label = l)) +
  geom_line(linetype = "dotted") +
  geom_point() +
  geom_text(position = position_nudge_line(x = 0.6))

ggplot(df, aes(x, y, label = l)) +
  geom_line(linetype = "dotted") +
  geom_point() +
  geom_text(position = position_nudge_line(y = 3.2))

ggplot(df, aes(x, y, label = l)) +
  geom_line(linetype = "dotted") +
  geom_point() +
  geom_text(position = position_nudge_line(x = 0.6, y = 3.2))

ggplot(df, aes(x, y, label = l)) +
  geom_line(linetype = "dotted") +
  geom_point() +
  geom_text(position = position_nudge_line(x = -0.6, y = -4))

# Other curves, using defaults

ggplot(df, aes(x, -y, label = l)) +
  geom_line(linetype = "dotted") +
  geom_point() +
  geom_text(position = position_nudge_line())

ggplot(df, aes(x, y - 40, label = l)) +
  geom_line(linetype = "dotted") +
  geom_point() +
  geom_text(position = position_nudge_line())

ggplot(subset(df, x >= 0), aes(y, sqrt(y), label = l)) +
  geom_line(linetype = "dotted") +
  geom_point() +
  geom_text(position = position_nudge_line())

# nudging outwards and downwards from a curve

ggplot(subset(df, x >= 0), aes(y, sqrt(y), label = l)) +
  geom_line(linetype = "dotted") +
  geom_point() +
  geom_text(position = position_nudge_line(xy_relative = -0.03))

# an arbitrary straight line

ggplot(df, aes(x, x * 2 + 5, label = l)) +
  geom_abline(intercept = 5, slope = 2, linetype = "dotted") +
  geom_point() +
```

```
  geom_text(position = position_nudge_line(abline = c(5, 2)))

# Points scattered near a curve or line, we use 'direction = "split"'

ggplot(subset(df, x >= 0), aes(x, yyy)) +
  stat_smooth(method = "lm", formula = y ~ x) +
  geom_point() +
  geom_text(aes(label = l),
            position = position_nudge_line(direction = "split"))

ggplot(df, aes(x)) +
  geom_line(aes(y = y), linetype = "dotted") +
  geom_point(aes(y = yy)) +
  geom_text(aes(y = yy, label = l),
            position = position_nudge_line(direction = "split"))

ggplot(subset(df, x >= 0), aes(y, yy)) +
  stat_smooth(method = "lm", formula = y ~ x) +
  geom_point() +
  geom_text(aes(label = l),
            position = position_nudge_line(direction = "split"))

# increasing the nudging for labels near the line

ggplot(subset(df, x >= 0), aes(y, yy)) +
  stat_smooth(method = "lm", formula = y ~ x) +
  geom_point() +
  geom_text(aes(label = l),
            position = position_nudge_line(line_nudge = 2,
                                           direction = "split"))

# fitting a linear model instead of the default spline

ggplot(subset(df, x >= 0), aes(y, yy)) +
  stat_smooth(method = "lm", formula = y ~ x) +
  geom_point() +
  geom_text(aes(label = l),
            position = position_nudge_line(method = "lm",
                                           direction = "split"))

ggplot(subset(df, x >= 0), aes(x, x^2)) +
  stat_smooth(method = "lm", formula = y ~ poly(x, 2, raw = TRUE)) +
  geom_point() +
  geom_text(aes(label = l),
            position = position_nudge_line(method = "lm",
                                           formula = y ~ poly(x, 2, raw = TRUE)))

ggplot(subset(df, x >= 0), aes(x, x^2)) +
  stat_smooth(method = "lm", formula = y ~ x + I(x^2)) +
  geom_point() +
  geom_text(aes(label = l),
            position = position_nudge_line(method = "lm",
                                           formula = y ~ x + I(x^2)))
```

```
# grouping is supported

df <- data.frame(x = rep(1:10, 2),
                 y = c(1:10, 10:1),
                 group = rep(c("a", "b"), c(10, 10)),
                 l = "+")

ggplot(df, aes(x, y, label = l, color = group)) +
  geom_line(linetype = "dotted") +
  geom_text() +
  geom_text(position = position_nudge_line()) +
  geom_text(position = position_nudge_line(xy_relative = -0.03))

# one needs to ensure that grouping is in effect in the geoms with nudging

ggplot(df, aes(x, y, label = l, color = group, group = group)) +
  geom_line(linetype = "dotted") +
  geom_text() +
  geom_text(color = "red",
            position = position_nudge_line()) +
  geom_text(color = "blue",
            position = position_nudge_line(xy_relative = -0.03)) +
  coord_equal()

# facets are also supported

ggplot(df, aes(x, y, label = l)) +
  geom_line(linetype = "dotted") +
  geom_text() +
  geom_text(position = position_nudge_line(xy_relative = c(0.06, 0.03)),
            color = "red") +
  geom_text(position = position_nudge_line(xy_relative = -c(0.06, 0.03)),
            color = "blue") +
  facet_wrap(~group) +
  coord_equal(ratio = 1.5)
```

---

position_nudge_to *Nudge labels to new positions*

Description

'position_nudge_to()' is generally useful for adjusting the position of labels or text, both on a discrete or continuous scale. This version from package 'ggpmisc' differs from [ggplot2::position_nudge] in that the coordinates of the new position is given directly, rather than as a displacement from the original location. As other position functions in this package, it preserves the original position to allow the text to be linked back to its original position with a segment or arrow.

Usage

```
position_nudge_to(x = NULL, y = NULL)
```

Arguments

x, y	Coordinates of the destination position. A numeric vector of length 1, or of the same length as rows there are in 'data'. The default, 'NULL', leaves the original coordinates unchanged.

Details

The new 'x' or 'y' replace the original ones, while the original coordinates are returned in 'x_orig' and 'y_orig'.

See Also

[ggplot::position_nudge()], [ggrepel::position_nudge_repel()].

Other position adjustments: `position_nudge_center()`, `position_nudge_line()`

Examples

```
df <- data.frame(
  x = c(1,3,2,5,4,2.5),
  y = c(2, 1, 2.5, 1.8, 2.8, 1.5),
  label = c("abc","cd","d","c","bcd","a")
)

ggplot(df, aes(x, y, label = label)) +
  geom_point() +
  geom_text(position = position_nudge_to(y = 3)
  )

ggplot(df, aes(x, y, label = label)) +
  geom_point() +
  geom_linked_text(position = position_nudge_to(y = 3),
                   vjust = -0.2)
```

---

quadrant_example.df *Example gene expression data*

Description

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

Usage

```
quadrant_example.df
```

Format

A `data.frame` object with 6088 rows and 6 variables

See Also

Other Transcriptomics data examples: `volcano_example.df`

Examples

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

`scale_colour_outcome` *Colour and fill scales for ternary outcomes*

Description

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

Usage

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

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

scale_fill_outcome(
  ...,
```

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

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

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

ggplot(my.df, aes(x, y, colour = outcome2)) +
  geom_point() +
  scale_colour_outcome() +
  theme_bw()
```

```
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

```
scale_npcx_continuous(...)

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

```
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

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

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

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

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

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

ggplot(my.df, aes(x, y, shape = outcome3, fill = outcome2)) +
```

```
  geom_point() +
  scale_shape_outcome(name = "direction") +
  scale_fill_outcome(name = "significance") +
  theme_bw()
```

`scale_x_logFC` *Position scales for log fold change data*

Description

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

Usage

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

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

Arguments

`name`	The name of the scale without units, used for the axis-label.
`breaks`	The positions of ticks or a function to generate them. Default varies depending on argument passed to `log.base.labels`. if supplied as a numeric vector they should be given using the data as passed to parameter `data`.

labels	The tick labels or a function to generate them from the tick positions. The default is function that uses the arguments passed to log.base.data and log.base.labels to generate suitable labels.
limits	limits One of: NULL to use the default scale range from ggplot2. A numeric vector of length two providing limits of the scale, using NA to refer to the existing minimum or maximum. A function that accepts the existing (automatic) limits and returns new limits. The default is function symmetric_limits() which keep 1 at the middle of the axis..
oob	Function that handles limits outside of the scale limits (out of bounds). The default squishes out-of-bounds values to the boundary.
expand	Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. The default is to expand the scale by 15% on each end for log-fold-data, so as to leave space for counts annotations.
log.base.labels, log.base.data	integer or logical Base of logarithms used to express fold-change values in tick labels and in data. Use FALSE for no logarithm transformation.
...	other named arguments passed to scale_y_continuous.

Details

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

See Also

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

Examples

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

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

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

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

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

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

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

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()))

# override "special" default arguments.
ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC("A concentration",
                breaks = waiver(),
                labels = waiver()) +
  scale_y_logFC("B concentration",
                breaks = waiver(),
                labels = waiver())

ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC() +
  scale_y_logFC() +
  geom_quadrant_lines() +
  stat_quadrant_counts(size = 3.5)
```

---

scale_y_Pvalue *Covenience scale for P-values*

Description

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

Usage

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

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

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

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

)

Arguments

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

```
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

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

```
  ...
)

stat_summary_xy(
  mapping = NULL,
  data = NULL,
  geom = "point",
  .fun.x = function(x) {     x },
  .fun.x.args = list(),
  .fun.y = function(y) {     y },
  .fun.y.args = list(),
  position = "identity",
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  ...
)

stat_centroid(
  mapping = NULL,
  data = NULL,
  geom = "point",
  .fun = mean,
  .fun.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, .fun`	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, .fun.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 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

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

Answers question "R ggplot on-the-fly calculation by grouping variable" at https://stackoverflow.com/questions/51412522.

Examples

```
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)
```

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

# 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))})

# Centroid
ggplot(my.df, aes(x = X, y = Y, colour = category)) +
  stat_centroid(shape = "cross", size = 6) +
  geom_point()

# Centroid
ggplot(my.df, aes(x = X, y = Y, colour = category)) +
  stat_centroid(geom = "text", aes(label = category)) +
  geom_point()
```

`stat_dens1d_filter` *Filter observations by local 1D density*

Description

`stat_dens1d_filter` Filters-out/filters-in observations in regions of a plot panel with high density of observations, based on the values mapped to one of `x` and `y` aesthetics. `stat_dens1d_filter_g` does the same 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

```
stat_dens1d_filter(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
```

```
  keep.fraction = 0.1,
  keep.number = Inf,
  keep.sparse = TRUE,
  invert.selection = FALSE,
  bw = "SJ",
  kernel = "gaussian",
  adjust = 1,
  n = 512,
  orientation = "x",
  na.rm = TRUE,
  show.legend = FALSE,
  inherit.aes = TRUE
)

stat_dens1d_filter_g(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  keep.fraction = 0.1,
  keep.number = Inf,
  keep.sparse = TRUE,
  invert.selection = FALSE,
  na.rm = TRUE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  bw = "SJ",
  adjust = 1,
  kernel = "gaussian",
  n = 512,
  orientation = "x",
  ...
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with aes or aes_. Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
keep.fraction	numeric [0..1]. The fraction of the observations (or rows) in data to be retained.
keep.number	integer Set the maximum number of observations to retain, effective only if obeying keep.fraction would result in a larger number.

keep.sparse — logical If TRUE, the default, observations from the more sparse regions are retained, if FALSE those from the densest regions.

invert.selection — logical If TRUE, the complement of the selected rows are returned.

bw — numeric or character The smoothing bandwidth to be used. If numeric, the standard deviation of the smoothing kernel. If character, a rule to choose the bandwidth, as listed in bw.nrd.

kernel — character See density for details.

adjust — numeric A multiplicative bandwidth adjustment. This makes it possible to adjust the bandwidth while still using the a bandwidth estimator through an argument passed to bw. The larger the value passed to adjust the stronger the smoothing, hence decreasing sensitivity to local changes in density.

n — numeric Number of equally spaced points at which the density is to be estimated for applying the cut point. See density for details.

orientation — character The aesthetic along which density is computed. Given explicitly by setting orientation to either "x" or "y".

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.

Value

A copy of data with a subset of the rows retained based on the filtering criterion.

See Also

density used internally.

Other statistics returning a subset of data: stat_dens1d_labels(), stat_dens2d_filter(), stat_dens2d_labels()

Examples

```
library(ggrepel)

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() })
)
d$xg <- d$x
d$xg[51:100] <- d$xg[51:100] + 1

# highlight the 1/10 of observations in sparsest regions of the plot
ggplot(data = d, aes(x, y)) +
  geom_point() +
  geom_rug(sides = "b") +
  stat_dens1d_filter(colour = "red") +
  stat_dens1d_filter(geom = "rug", colour = "red", sides = "b")

# highlight the 1/4 of observations in densest regions of the plot
ggplot(data = d, aes(x, y)) +
  geom_point() +
  geom_rug(sides = "b") +
  stat_dens1d_filter(colour = "blue",
                     keep.fraction = 1/4, keep.sparse = FALSE) +
  stat_dens1d_filter(geom = "rug", colour = "blue",
                     keep.fraction = 1/4, keep.sparse = FALSE,
                     sides = "b")

# switching axes
ggplot(data = d, aes(x, y)) +
  geom_point() +
  geom_rug(sides = "l") +
  stat_dens1d_filter(colour = "red", orientation = "y") +
  stat_dens1d_filter(geom = "rug", colour = "red", orientation = "y",
                     sides = "l")

# highlight 1/10 plus 1/10 observations in high and low density regions
ggplot(data = d, aes(x, y)) +
  geom_point() +
  geom_rug(sides = "b") +
  stat_dens1d_filter(colour = "red") +
  stat_dens1d_filter(geom = "rug", colour = "red", sides = "b") +
  stat_dens1d_filter(colour = "blue", keep.sparse = FALSE) +
  stat_dens1d_filter(geom = "rug",
                     colour = "blue", keep.sparse = FALSE, sides = "b")

# selecting the 1/10 observations in sparsest regions and their complement
ggplot(data = d, aes(x, y)) +
  stat_dens1d_filter(colour = "red") +
  stat_dens1d_filter(geom = "rug", colour = "red", sides = "b") +
  stat_dens1d_filter(colour = "blue", invert.selection = TRUE) +
  stat_dens1d_filter(geom = "rug",
                     colour = "blue", invert.selection = TRUE, sides = "b")

# density filtering done jointly across groups
ggplot(data = d, aes(xg, y, colour = group)) +
  geom_point() +
  geom_rug(sides = "b", colour = "black") +
```

```
  stat_dens1d_filter(shape = 1, size = 3, keep.fraction = 1/4, adjust = 2)

# density filtering done independently for each group
ggplot(data = d, aes(xg, y, colour = group)) +
  geom_point() +
  geom_rug(sides = "b") +
  stat_dens1d_filter_g(shape = 1, size = 3, keep.fraction = 1/4, adjust = 2)

# density filtering done jointly across groups by overriding grouping
ggplot(data = d, aes(xg, y, colour = group)) +
  geom_point() +
  geom_rug(sides = "b") +
  stat_dens1d_filter_g(colour = "black",
                       shape = 1, size = 3, keep.fraction = 1/4, adjust = 2)

# label observations
ggplot(data = d, aes(x, y, label = lab, colour = group)) +
  geom_point() +
  stat_dens1d_filter(geom = "text", hjust = "outward")

# repulsive labels with ggrepel::geom_text_repel()
ggplot(data = d, aes(x, y, label = lab, colour = group)) +
  geom_point() +
  stat_dens1d_filter(geom = "text_repel")
```

`stat_dens1d_labels` *Replace labels in data based on 1D density*

Description

`stat_dens1d_labels()` Sets values mapped to the `label` aesthetic to `""` or a user provided character string based on the local density in regions of a plot panel. Its main use is together with repulsive geoms from package `ggrepel`. If there is no mapping to `label` in `data`, the mapping is set to `rownames(data)`, with a message.

Usage

```
stat_dens1d_labels(
  mapping = NULL,
  data = NULL,
  geom = "text",
  position = "identity",
  ...,
  keep.fraction = 0.1,
  keep.number = Inf,
  keep.sparse = TRUE,
  invert.selection = FALSE,
  bw = "SJ",
```

```
  kernel = "gaussian",
  adjust = 1,
  n = 512,
  orientation = "x",
  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
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
keep.fraction	numeric [0..1]. The fraction of the observations (or rows) in data to be retained.
keep.number	integer Set the maximum number of observations to retain, effective only if obeying keep.fraction would result in a larger number.
keep.sparse	logical If TRUE, the default, observations from the more sparse regions are retained, if FALSE those from the densest regions.
invert.selection	logical If TRUE, the complement of the selected rows are returned.
bw	numeric or character The smoothing bandwidth to be used. If numeric, the standard deviation of the smoothing kernel. If character, a rule to choose the bandwidth, as listed in bw.nrd.
kernel	character See density for details.
adjust	numeric A multiplicative bandwidth adjustment. This makes it possible to adjust the bandwidth while still using the a bandwidth estimator through an argument passed to bw. The larger the value passed to adjust the stronger the smoothing, hence decreasing sensitivity to local changes in density.
n	numeric Number of equally spaced points at which the density is to be estimated for applying the cut point. See density for details.
orientation	character The aesthetic along which density is computed. Given explicitly by setting orientation to either "x" or "y".
label.fill	character vector of length 1 or a function.
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.

Details

`stat_dens1d_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, `""`. This makes `stat_dens1d_filter` unsuitable for the task. Non-the-less `stat_dens1d_labels()` could be useful in some other cases, as the substitution character string can be set by the user.

Value

A copy of `data` with a subset of the rows retained based on the filtering criterion.

See Also

`density` used internally.

Other statistics returning a subset of data: `stat_dens1d_filter()`, `stat_dens2d_filter()`, `stat_dens2d_labels()`

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() })
)

# using defaults
ggplot(data = d, aes(x, y, label = lab)) +
  geom_point() +
  stat_dens1d_labels()

# using defaults
ggplot(data = d, aes(x, y, label = lab)) +
  geom_point() +
  stat_dens1d_labels(geom = "text_repel")

# if no mapping to label is found, it is set row names
ggplot(data = d, aes(x, y)) +
  geom_point() +
  stat_dens1d_labels(geom = "text_repel")

# using defaults, along y-axis
```

```
ggplot(data = d, aes(x, y, label = lab)) +
  geom_point() +
  stat_dens1d_labels(orientation = "y", geom = "text_repel")

# example labelling with coordiantes
ggplot(data = d, aes(x, y, label = sprintf("x = %.2f\ny = %.2f", x, y))) +
  geom_point() +
  stat_dens1d_filter(colour = "red") +
  stat_dens1d_labels(geom = "text_repel", colour = "red", size = 3)

# 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_dens1d_labels(geom = "debug")

ggplot(data = d, aes(x, y, label = lab, colour = group)) +
  geom_point() +
  stat_dens1d_labels(geom = "text_repel")

ggplot(data = d, aes(x, y, label = lab, colour = group)) +
  geom_point() +
  stat_dens1d_labels(geom = "text_repel", label.fill = NA)

# we keep labels starting with "a" across the whole plot, but all in sparse
# regions. To achieve this we pass as argument to label.fill a fucntion
# instead of a character string.
label.fun <- function(x) {ifelse(grepl("^a", x), x, "")}
ggplot(data = d, aes(x, y, label = lab, colour = group)) +
  geom_point() +
  stat_dens1d_labels(geom = "text_repel", label.fill = label.fun)
```

`stat_dens2d_filter` *Filter observations by local 2D density*

Description

`stat_dens2d_filter` Filters-out/filters-in observations in regions of a plot panel with high density of observations, based on the values mapped to both `x` and `y` aesthetics. `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

```
stat_dens2d_filter(
  mapping = NULL,
  data = NULL,
```

```
  geom = "point",
  position = "identity",
  keep.fraction = 0.1,
  keep.number = Inf,
  keep.sparse = TRUE,
  invert.selection = FALSE,
  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,
  invert.selection = FALSE,
  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]. The fraction of the observations (or rows) in data to be retained.
keep.number	integer Set the maximum number of observations to retain, effective only if obeying keep.fraction would result in a larger number.
keep.sparse	logical If TRUE, the default, observations from the more sparse regions are retained, if FALSE those from the densest regions.
invert.selection	logical If TRUE, the complement of the selected rows are returned.
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.

Value

A copy of data with a subset of the rows retained based on a 2D-density-based filtering criterion.

See Also

kde2d used internally.

Other statistics returning a subset of data: stat_dens1d_filter(), stat_dens1d_labels(), stat_dens2d_labels()

Examples

```
library(ggrepel)

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() })
)

# filter (and here highlight) 1/10 observations in sparsest regions
ggplot(data = d, aes(x, y)) +
  geom_point() +
  stat_dens2d_filter(colour = "red")

# filter observations not in the sparsest regions
ggplot(data = d, aes(x, y)) +
  geom_point() +
  stat_dens2d_filter(colour = "blue", invert.selection = TRUE)
```

```
# filter observations in dense regions of the plot
ggplot(data = d, aes(x, y)) +
  geom_point() +
  stat_dens2d_filter(colour = "blue", keep.sparse = FALSE)

# filter 1/2 the observations
ggplot(data = d, aes(x, y)) +
  geom_point() +
  stat_dens2d_filter(colour = "red", keep.fraction = 0.5)

# filter 1/2 the observations but cap their number to maximum 12 observations
ggplot(data = d, aes(x, y)) +
  geom_point() +
  stat_dens2d_filter(colour = "red",
                     keep.fraction = 0.5,
                     keep.number = 12)

# density filtering done jointly across groups
ggplot(data = d, aes(x, y, colour = group)) +
  geom_point() +
  stat_dens2d_filter(shape = 1, size = 3, keep.fraction = 1/4)

# density filtering done independently for each group
ggplot(data = d, aes(x, y, colour = group)) +
  geom_point() +
  stat_dens2d_filter_g(shape = 1, size = 3, keep.fraction = 1/4)

# density filtering done jointly across groups by overriding grouping
ggplot(data = d, aes(x, y, colour = group)) +
  geom_point() +
  stat_dens2d_filter_g(colour = "black",
                       shape = 1, size = 3, keep.fraction = 1/4)

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

# repulsive labels with ggrepel::geom_text_repel()
ggplot(data = d, aes(x, y, label = lab, colour = group)) +
  geom_point() +
  stat_dens2d_filter(geom = "text_repel")
```

`stat_dens2d_labels` *Replace labels in data based on 2D density*

Description

`stat_dens2d_labels()` Sets values mapped to the `label` aesthetic to `""` or a user provided character string based on the local density in regions of a plot panel. Its main use is together with

repulsive geoms from package ggrepel. If there is no mapping to label in data, the mapping is set to rownames(data), with a message.

Usage

```
stat_dens2d_labels(
  mapping = NULL,
  data = NULL,
  geom = "text",
  position = "identity",
  ...,
  keep.fraction = 0.1,
  keep.number = Inf,
  keep.sparse = TRUE,
  invert.selection = FALSE,
  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
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
keep.fraction	numeric [0..1]. The fraction of the observations (or rows) in data to be retained.
keep.number	integer Set the maximum number of observations to retain, effective only if obeying keep.fraction would result in a larger number.
keep.sparse	logical If TRUE, the default, observations from the more sparse regions are retained, if FALSE those from the densest regions.
invert.selection	logical If TRUE, the complement of the selected rows are returned.
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 vector of length 1 or a function.

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.

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, "". This makes stat_dens2d_filter unsuitable for the task. Non-the-less stat_dens2d_labels() could be useful in some other cases, as the substitution character string can be set by the user.

Value

A copy of data with none, some or possibly all values for aesthetic label replaced by the value of label.fill based on a 2D-density-based filtering criterion.

See Also

kde2d used internally.

Other statistics returning a subset of data: stat_dens1d_filter(), stat_dens1d_labels(), 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() })
)

# using defaults
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, colour = group)) +
  geom_point() +
  stat_dens2d_labels(geom = "text_repel")

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

# we keep labels starting with "a" across the whole plot, but all in sparse
# regions. To achieve this we pass as argument to label.fill a fucntion
# instead of a character string.
label.fun <- function(x) {ifelse(grepl("^a", x), x, "")}
ggplot(data = d, aes(x, y, label = lab, colour = group)) +
  geom_point() +
  stat_dens2d_labels(geom = "text_repel", label.fill = label.fun)
```

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

Description

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

Usage

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

Arguments

mapping	The aesthetic mapping, usually constructed with aes or aes_. Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
method	character or function.
method.args, augment.args	list of arguments to pass to method and to 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 wich 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 intead of the original variable names, while data is automatically passed the data frame. This helps ensure that the model is fitted to the same data as plotted in other layers.

Warning!

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

Handling of grouping

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

Computed variables

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

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

Note

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

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

See Also

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

Other Statistics calling generic tidier methods.: `stat_fit_glance()`, `stat_fit_tb()`, `stat_fit_tidy()`

Examples

```
library(broom)
library(gginnards)
library(quantreg)

# 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.args = list(formula = y ~ x),
                   y.out = ".resid")

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

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

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

`stat_fit_deviations` *Residuals from model fit as segments*

Description

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

Usage

```
stat_fit_deviations(
  mapping = NULL,
  data = NULL,
  geom = "segment",
  method = "lm",
```

```
  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 intead 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()

Examples

```
library(gginnards) # needed for geom_debug()
# 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)) +
  geom_smooth(method = "lm", formula = my.formula) +
  stat_fit_deviations(formula = my.formula, colour = "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, colour = "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 "tidy" version of the model's fit, using 'glance() methods from packages 'broom', 'broom.mixed', or other sources.

Usage

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

Arguments

mapping	The aesthetic mapping, usually constructed with aes or aes_. Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific data set - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
method	character or function.
method.args, glance.args	list of arguments to pass to method and to [generics::glance()], respectively.
label.x, label.y	numeric with range 0..1 "normalized parent coordinates" (npc units) or character if using geom_text_npc() or geom_label_npc(). If using geom_text() or geom_label() numeric in native data units. If too short they will be recycled.
hstep, vstep	numeric in npc units, the horizontal and vertical step used between labels for different groups.
position	The position adjustment to use for overlapping points on this layer
na.rm	a logical indicating whether NA values should be stripped before the computation proceeds.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders.
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.

Details

stat_fit_glance together with stat_fit_tidy and stat_fit_augment, based on package 'broom' can be used with a broad range of model fitting functions as supported at any given time by package 'broom'. In contrast to stat_poly_eq wich can generate text or expression labels automatically, for these functions the mapping of aesthetic label needs to be explicitly supplied in the callm, 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 intead of the original variable names, while data is automatically passed the data frame. This helps ensure that the model is fitted to the same data as plotted in other layers.

Warning!

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

Handling of grouping

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

Model formula required

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

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 consitent 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.

Note

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

See Also

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

Other Statistics calling generic tidier methods.: stat_fit_augment(), stat_fit_tb(), stat_fit_tidy()

Examples

```
library(broom)
library(gginnards)
library(quantreg)

# 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))),
                  parse = TRUE)

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

# correlation test
```

```
ggplot(mtcars, aes(x = disp, y = mpg)) +
  geom_point() +
  stat_fit_glance(method = "cor.test",
                  label.y = "bottom",
                  method.args = list(formula = ~ x + y),
                 mapping = aes(label = sprintf('r[Pearson]~"="~%.3f~~italic(P)~"="~%.2g',
                                stat(estimate), stat(p.value))),
                  parse = TRUE)

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

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

`stat_fit_residuals` *Residuals from a model fit*

Description

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

Usage

```
stat_fit_residuals(
  mapping = NULL,
  data = NULL,
  geom = "point",
  method = "lm",
  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 intead of the original variable names, while data is automatically passed the data frame. This helps ensure that the model is fitted to the same data as plotted in other layers.

Computed variables

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

x x coordinates of observations

y.resid residuals from fitted values

y.resid.abs absolute residuals from the fit

.

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

See Also

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

Examples

```
# generate artificial data
set.seed(4321)
x <- 1:100
y <- (x + x^2 + x^3) + rnorm(length(x), mean = 0, sd = mean(x^3) / 4)
my.data <- data.frame(x, 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 'tidy() methods from packages 'broom', 'broom.mixed', or other sources. The annotation is added to the plots in tabular form.

Usage

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

```
  position = "identity",
  table.theme = NULL,
  table.rownames = FALSE,
  table.colnames = TRUE,
  table.hjust = 1,
  parse = FALSE,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with aes or aes_. Only needs to be set at the layer level if you are overriding the plot defaults.
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, tidy.args	lists of arguments to pass to method and to tidy().
tb.type	character One of "fit.summary", "fit.anova" or "fit.coefs".
tb.vars, tb.params	character or numeric vectors, optionally named, used to select and/or rename the columns or the parameters in the table returned.
digits	integer indicating the number of significant digits to be used for all numeric values in the table.
p.digits	integer indicating the number of decimal places to round p-values to, with those rounded to zero displayed as the next larger possible value preceded by "<". If p.digits is outside the range 1..22 no rounding takes place.
label.x, label.y	numeric Coordinates (in data units) to be used for absolute positioning of the output. If too short they will be recycled.
label.x.npc, label.y.npc	numeric with range 0..1 or character. Coordinates to be used for positioning the output, expressed in "normalized parent coordinates" or character string. If too short they will be recycled.
position	The position adjustment to use for overlapping points on this layer
table.theme	NULL, list or function A gridExtra ttheme defintion, or a constructor for a ttheme or NULL for default.
table.rownames, table.colnames	logical flag to enable or disabling printing of row names and column names.
table.hjust	numeric Horizontal justification for the core and column headings of the table.
parse	If TRUE, the labels will be parsed into expressions and displayed as described in ?plotmath.

na.rm	a logical indicating whether NA values should be stripped before the computation proceeds.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders.
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.

Details

stat_fit_tb Applies a model fitting function per panel, using the grouping factors from easthetic 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 orverridden.

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

Other Statistics calling generic tidier methods.: stat_fit_augment(), stat_fit_glance(), stat_fit_tidy()

Examples

```
library(broom)

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

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

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

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

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

# Linear regression fit coeficients
ggplot(my.df, aes(covariate, x)) +
  geom_point() +
  stat_fit_tb(tb.type = "fit.coefs") +
  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)

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

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

```
             parse = TRUE) +
  expand_limits(y = 70)

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

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

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

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

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

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

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

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

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

`stat_fit_tidy` *One row data frame with fitted parameter estimates*

Description

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

Usage

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

Arguments

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

data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
method	character or function.
method.args, tidy.args	list of arguments to pass to method, and to [generics::tidy], respectively.
label.x, label.y	numeric with range 0..1 or character. Coordinates to be used for positioning the output, expressed in "normalized parent coordinates" or character string. If too short they will be recycled.
hstep, vstep	numeric in npc units, the horizontal and vertical step used between labels for different groups.
position	The position adjustment to use for overlapping points on this layer
na.rm	a logical indicating whether NA values should be stripped before the computation proceeds.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders.
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.

Details

stat_fit_tidy together with stat_fit_glance and stat_fit_augment, based on package 'broom' can be used with a broad range of model fitting functions as supported at any given time by 'broom'. In contrast to stat_poly_eq wich 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 intead of the original variable names, while data is automatically passed the data frame. This helps ensure that the model is fitted to the same data as plotted in other layers.

Warning!

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

Handling of grouping

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

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

See Also

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

Other Statistics calling generic tidier methods.: `stat_fit_augment()`, `stat_fit_glance()`, `stat_fit_tb()`

Examples

```
library(broom)
library(gginnards)
library(quantreg)

# 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\np-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\np-value = %.3g",
                                              stat(x_estimate),
                                              stat(x_p.value))))

# Correlation test
ggplot(mtcars, aes(x = disp, y = mpg)) +
  stat_smooth(method = "lm") +
  geom_point() +
  stat_fit_tidy(method = "cor.test",
                label.y = "bottom",
                method.args = list(formula = ~ x + y),
                mapping = aes(label = sprintf("R = %.3g\np-value = %.3g",
                                              stat(`_estimate`),
                                              stat(`_p.value`))))

# Quantile regression
ggplot(mtcars, aes(x = disp, y = mpg)) +
  stat_smooth(method = "lm") +
  geom_point() +
  stat_fit_tidy(method = "rq",
                label.y = "bottom",
                method.args = list(formula = y ~ x),
                tidy.args = list(se.type = "nid"),
                mapping = aes(label = sprintf("Slope = %.3g\np-value = %.3g",
                                              stat(x_estimate),
                                              stat(x_p.value))))
```

---

stat_fmt_tb *Select and slice a tibble nested in* data

Description

stat_fmt_tb selects, reorders and/or renames columns and or rows of a tibble nested in data. This stat is intended 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",
  table.theme = NULL,
  table.rownames = FALSE,
  table.colnames = TRUE,
  table.hjust = 0.5,
  parse = FALSE,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with aes or aes_. Only needs to be set at the layer level if you are overriding the plot defaults.
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, tb.rows	character or numeric vectors, optionally named, used to select and/or rename the columns or rows in the table returned.
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
table.theme	NULL, list or function A gridExtra ttheme definition, or a constructor for a ttheme or NULL for default.
table.rownames, table.colnames	logical flag to enable or disabling printing of row names and column names.
table.hjust	numeric Horizontal justification for the core and column headings of the table.
parse	If TRUE, the labels will be parsed into expressions and displayed as described in ?plotmath.
na.rm	a logical indicating whether NA values should be stripped before the computation proceeds.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders.

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

Value

The returned value is a copy codedata in which the data frames mapped to label have been modified.

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.

See Also

See geom_table for details on how tables respond to mapped aesthetics and table themes. For details on predefined table themes see ttheme_gtdefault.

Examples

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

# Hide column names, diplay row names
ggplot(my.df, aes(x, y, label = tbs)) +
  stat_fmt_tb(table.colnames = FALSE,
              table.rownames = TRUE) +
  expand_limits(x = c(0,3), y = c(-2, 6))

# Use a theme for the table
ggplot(my.df, aes(x, y, label = tbs)) +
  stat_fmt_tb(table.theme = ttheme_gtlight) +
  expand_limits(x = c(0,3), y = c(-2, 6))

# selection and renaming by column position
ggplot(my.df, aes(x, y, label = tbs)) +
  stat_fmt_tb(tb.vars = c(value = 1, group = 2),
              tb.rows = 1:3) +
  expand_limits(x = c(0,3), y = c(-2, 6))

# selection, reordering and renaming by column position
ggplot(my.df, aes(x, y, label = tbs)) +
```

```
  stat_fmt_tb(tb.vars = c(group = 2, value = 1),
              tb.rows = 1:3) +
  expand_limits(x = c(0,3), y = c(-2, 6))

# selection and renaming, using partial matching to column name
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 a subset of data with rows matching for peaks or valleys with formatted character labels added. The formatting is determined by a format string compatible with sprintf() or strftime().

Usage

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

stat_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 strptime, 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.

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

Returned and computed variables

x x-value at the peak (or valley) as numeric

y y-value at the peak (or valley) as numeric

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

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

Note

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

See Also

Other peaks and valleys functions: find_peaks()

Examples

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

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

```
  stat_peaks(colour = "red") +
  stat_peaks(colour = "red", geom = "text", hjust = -0.1, angle = 33)

lynx_datetime.df <-
   try_tibble(lynx,
              col.names = c("year", "lynx")) # years -> POSIXct
ggplot(lynx_datetime.df, aes(year, lynx)) +
  geom_line() +
  stat_peaks(colour = "red") +
  stat_valleys(colour = "blue")
ggplot(lynx_datetime.df, aes(year, lynx)) +
  geom_line() +
  stat_peaks(colour = "red") +
  stat_peaks(colour = "red",
             geom = "text",
             hjust = -0.1,
             x.label.fmt = "%Y",
             angle = 33)
```

---

stat_poly_eq *Equation, p-value, R^2, AIC or BIC of fitted polynomial*

Description

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

Usage

```
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,
  f.digits = 3,
  p.digits = 3,
  label.x = "left",
  label.y = "top",
  label.x.npc = NULL,
  label.y.npc = NULL,
  hstep = 0,
```

```
  vstep = NULL,
  output.type = "expression",
  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
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
formula	a formula object. Using aesthetic names instead of original variable names.
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, f.digits	integer Number of significant digits to use for the fitted coefficients and F-value.
rr.digits, p.digits	integer Number of digits after the decimal point to use for R^2 and P-value in 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", "text", "markdown" 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 undertood 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 numric 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

f.value.label F value and degrees of freedom for the fitted model as a whole.

p.value..label P-value for the F-value above.

AIC.label AIC for the fitted model.

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, f.value, f.df1, f.df2, p.value, AIC, BIC numeric values extracted or computed 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

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 code for construction of character strings from numeric values and their mapping to aesthetic `label` needs to be explicitly supplied in the call.

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

Examples

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

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

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

ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(aes(label =  paste(stat(f.value.label),
                                  stat(p.value.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 = 4,
               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]~`=`~%.3g*\", \"*b[1]~`=`~%.3g*\", \"*b[2]~`=`~%.3g*\", \"*b[3]~`=`~%.3g"
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(formula = formula,
               output.type = "numeric",
               parse = TRUE,
               mapping =
                aes(label = sprintf(my.format,
                                    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(aes(label = stat(eq.label)),
               formula = formula, geom = "debug",
               output.type = "markdown")

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]]})
```

`stat_quadrant_counts` *Number of observations in quadrants*

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 en 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",
  position = "identity",
  quadrants = NULL,
  pool.along = "none",
  xintercept = 0,
  yintercept = 0,
  label.x = NULL,
  label.y = NULL,
  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
`position`	The position adjustment to use for overlapping points on this layer
`quadrants`	integer vector indicating which quadrants are of interest, with a `0L` 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 statistic 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 `xintercept` or `yintercept` are counted together with those larger than the intercepts. 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

```
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)))) +
  expand_limits(y = 12.7)

ggplot(my.data, aes(x, y)) +
  geom_quadrant_lines(colour = "blue", xintercept = 50, yintercept = 10) +
  stat_quadrant_counts(colour = "blue", xintercept = 50, yintercept = 10) +
  geom_point() +
  scale_y_continuous(expand = expansion(mult = 0.15, add = 0))

ggplot(my.data, aes(x, y)) +
  geom_quadrant_lines(colour = "blue",
                      pool.along = "x", yintercept = 10) +
  stat_quadrant_counts(colour = "blue", label.x = "right",
                      pool.along = "x", yintercept = 10) +
  geom_point() +
  expand_limits(y = c(7, 13))

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quadrant_counts(quadrants = 0, label.x = "left", label.y = "bottom")

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quadrant_counts(geom = "text") # use "tex" instead
```

symmetric_limits *Expand a range to make it symmetric*

Description

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

Usage

```
symmetric_limits(x)
```

Arguments

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

Value

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

Examples

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

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

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

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

```
class(lynx)
try_tibble(lynx)
try_tibble(lynx, as.numeric = TRUE)
try_tibble(lynx, "year")
class(austres)
try_tibble(austres)
try_tibble(austres, as.numeric = TRUE)
```

```
try_tibble(austres, "quarter")
class(cars)
try_tibble(cars)
```

`ttheme_gtdefault` *Table themes*

Description

Additional theme constructors for use with `geom_table`.

Usage

```
ttheme_gtdefault(
  base_size = 10,
  base_colour = "black",
  base_family = "",
  parse = FALSE,
  padding = unit(c(0.8, 0.6), "char"),
  ...
)

ttheme_gtminimal(
  base_size = 10,
  base_colour = "black",
  base_family = "",
  parse = FALSE,
  padding = unit(c(0.5, 0.4), "char"),
  ...
)

ttheme_gtbw(
  base_size = 10,
  base_colour = "black",
  base_family = "",
  parse = FALSE,
  padding = unit(c(0.8, 0.6), "char"),
  ...
)

ttheme_gtplain(
  base_size = 10,
  base_colour = "black",
  base_family = "",
  parse = FALSE,
  padding = unit(c(0.8, 0.6), "char"),
```

```
  ...
)

ttheme_gtdark(
  base_size = 10,
  base_colour = "grey90",
  base_family = "",
  parse = FALSE,
  padding = unit(c(0.8, 0.6), "char"),
  ...
)

ttheme_gtlight(
  base_size = 10,
  base_colour = "grey10",
  base_family = "",
  parse = FALSE,
  padding = unit(c(0.8, 0.6), "char"),
  ...
)

ttheme_gtsimple(
  base_size = 10,
  base_colour = "grey10",
  base_family = "",
  parse = FALSE,
  padding = unit(c(0.5, 0.4), "char"),
  ...
)

ttheme_gtstripes(
  base_size = 10,
  base_colour = "grey10",
  base_family = "",
  parse = FALSE,
  padding = unit(c(0.8, 0.6), "char"),
  ...
)
```

Arguments

base_size	numeric, default font size.
base_colour	default font colour.
base_family	default font family.
parse	logical, default behaviour for parsing text as plotmath.
padding	length-2 unit vector specifying the horizontal and vertical padding of text within each cell.
...	further arguments to control the gtable.

Details

Depending on the theme, the base_colour, which is mapped to the `colour` aesthetic if present, is applied to only the text elements, or to the text elements and rules. The difference is exemplified below.

Value

A `list` object that can be used as `ttheme` in the construction of tables with functions from package 'gridExtra'.

Note

These theme constructors are wrappers on `gridExtra::ttheme_default()` and `gridExtra::ttheme_minimal()`. They can also be used with `grid.table` if desired.

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

df <- tibble(x = 5.45, y = 34, tb = list(tb))

# Same as the default theme constructor
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df, aes(x = x, y = y, label = tb),
             table.theme = ttheme_gtdefault) +
  theme_classic()

# Minimal theme constructor
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df, aes(x = x, y = y, label = tb),
             table.theme = ttheme_gtminimal) +
  theme_classic()

# A theme with white background
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df, aes(x = x, y = y, label = tb),
             table.theme = ttheme_gtbw) +
  theme_bw()

# Default colour of theme superceded by aesthetic constant
```

```
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df, aes(x = x, y = y, label = tb),
             table.theme = ttheme_gtbw, colour = "darkblue") +
  theme_bw()

# A theme with dark background
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df, aes(x = x, y = y, label = tb),
             table.theme = ttheme_gtdark) +
  theme_dark()

# Default colour of theme superceded by aesthetic constant
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df, aes(x = x, y = y, label = tb),
             table.theme = ttheme_gtdark, colour = "yellow") +
  theme_dark()

# A theme with light background
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df, aes(x = x, y = y, label = tb),
             table.theme = ttheme_gtlight)

# Default colour of theme superceded by aesthetic constant
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df, aes(x = x, y = y, label = tb),
             table.theme = ttheme_gtlight, colour = "darkred")

# Default colour of theme superceded by aesthetic constant
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df, aes(x = x, y = y, label = tb),
             table.theme = ttheme_gtsimple)

# Default colour of theme superceded by aesthetic constant
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df, aes(x = x, y = y, label = tb),
             table.theme = ttheme_gtstripes) +
  theme_dark()
```

---

ttheme_set *Set default table theme*

Description

Set R option to the theme to use as current default. This function is implemented differently but is used in the same way as `ggplot2::theme_set()` but affects the default table-theme instead of the plot theme.

Usage

```
ttheme_set(table.theme = NULL)
```

Arguments

`table.theme`	NULL, list or function A gridExtra ttheme defintion, or a constructor for a ttheme or NULL for default.

Value

A named list with the previous value of the option.

Note

The ttheme is set when a plot object is constructed, and consequently the option setting does not affect rendering of ready built plot objects.

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

df <- tibble(x = 5.45, y = 34, tb = list(tb))

# Same as the default theme constructor
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df, aes(x = x, y = y, label = tb))

# set a new default
old_ttheme <- ttheme_set(ttheme_gtstripes)

ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df, aes(x = x, y = y, label = tb))

# restore previous setting
ttheme_set(old_ttheme)
```

volcano_example.df *Example gene expression data*

Description

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

Usage

```
volcano_example.df
```

Format

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

See Also

Other Transcriptomics data examples: `quadrant_example.df`

Examples

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

xy_outcomes2factor *Convert two numeric ternary outcomes into a factor*

Description

Convert two numeric ternary outcomes into a factor

Usage

```
xy_outcomes2factor(x, y)

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

Arguments

`x, y`	numeric vectors of -1, 0, and +1 values, indicating down regulation, uncertain response or up-regulation, or numeric vectors that can be converted into such values using a pair of thresholds.
`x_threshold, y_threshold`	numeric vector Ranges enclosing the values to be considered uncertain for each of the two vectors..

Details

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

Note

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

See Also

Other Functions for quadrant and volcano plots: `FC_format()`, `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