Package 'ggpp'

May 8, 2026

Type Package

Title Grammar Extensions to 'ggplot2'

Version 0.6.0

Date 2026-01-18

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

Description Extensions to 'ggplot2' respecting the grammar of graphics paradigm. Geometries: geom_table(), geom_plot() and geom_grob() add insets to plots using native data coordinates, while geom_table_npc(), geom_plot_npc() and geom_grob_npc() do the same using ``npc" coordinates through new aesthetics ``npcx" and ``npcy". Statistics: select observations based on 2D density. Positions: radial nudging away from a center point and nudging away from a line or curve; combined stacking and nudging; combined dodging and nudging.

License GPL (>= 2)

LazyData TRUE

LazyLoad TRUE

ByteCompile TRUE

Depends R (>= 4.1.0), ggplot2 (>= 3.5.0)

Imports stats, grid, grDevices, rlang (>= 1.0.6), vctrs (>= 0.6.0), glue (>= 1.6.0), gridExtra (>= 2.3), scales (>= 1.3.0), tibble (>= 3.1.8), dplyr (>= 1.1.0), xts (>= 0.13.0), zoo (>= 1.8-11), MASS (>= 7.3-58), polynom (>= 1.4-0), lubridate (>= 1.9.0), stringr (>= 1.4.0)

Suggests knitr (>= 1.40), rmarkdown (>= 2.20), ggrepel (>= 0.9.2), gginnards(>= 0.2.0), magick (>= 2.7.3), testthat (>= 3.1.5), vdiffr (>= 1.0.5)

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

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

Encoding UTF-8

RoxygenNote 7.3.3

VignetteBuilder knitr

Collate 'annotate.r' 'compute-npc.r' 'dark-or-light.R'
'example-data.R' 'geom-grob.R' 'ggpp-legend-draw.R'
'utilities.R' 'ggp2-margins.R' 'geom-label-linked.r'
'geom-label-npc.r' 'geom-label-pairwise.r'
'geom-margin-arrow.r' 'geom-margin-grob.r'
'geom-margin-point.r' 'geom-plot.R' 'geom-point-linked.r'
'geom-quadrant-lines.R' 'geom-table.R' 'geom-text-linked.r'
'geom-text-npc.r' 'geom-text-pairwise.R' 'ggpp.R' 'onload.R'
'position-dodge-nudge-to.R' 'position-dodge-nudge.R'
'position-dodge2-nudge.R' 'position-dodge2nudge-to.R'
'position-jitter-nudge.R' 'position-nudge-center.R'
'position-nudge-line.R' 'position-stack-nudge.R'
'position-stacknudge-to.R' 'scale-continuous-npc.r'
'stat-apply.R' 'stat-dens1d-filter.r' 'stat-dens1d-labels.r'
'stat-dens2d-filter.r' 'stat-dens2d-labels.r'
'stat-format-table.R' 'stat-functions.R' 'stat-panel-counts.R'
'stat-quadrant-counts.R' 'try-data-frame.R' 'weather-data.R'
'wrap-labels.R'

Config/Needs/website rmarkdown

NeedsCompilation no

Author Pedro J. Aphalo [aut, cre] (ORCID:
<https://orcid.org/0000-0003-3385-972X>),
Kamil Slowikowski [ctb] (ORCID:
<https://orcid.org/0000-0002-2843-6370>),
Michał Krassowski [ctb] (ORCID:
<https://orcid.org/0000-0002-9638-7785>),
Daniel Sabanés Bové [ctb] (ORCID:
<https://orcid.org/0000-0002-0176-9239>),
Stella Banjo [ctb]

Repository CRAN

Date/Publication 2026-01-18 17:40:02 UTC

Contents

`ggpp-package` *ggpp: Grammar Extensions to 'ggplot2'*

Description

Extensions to 'ggplot2' respecting the grammar of graphics paradigm. Geometries: geom_table(), geom_plot() and geom_grob() add insets to plots using native data coordinates, while geom_table_npc(), geom_plot_npc() and geom_grob_npc() do the same using "npc" coordinates through new aesthetics "npcx" and "npcy". Statistics: select observations based on 2D density. Positions: radial nudging away from a center point and nudging away from a line or curve; combined stacking and nudging; combined dodging and nudging.

Details

Package 'ggpp' provides functions that extend the grammar of graphics as implemented in 'ggplot2'. It attempts to stay true to the original grammar and to respect the naming conventions used in 'ggplot2'.

Extensions provided:

- Geoms adding support for plot, table and grob insets within the gramamr. Geoms using a parallel pseudo-scale based on native plot coordinates (npc) to allow annotations consistent with the grammar and so supporting facets and grouping. Geoms for annotations on the edges of the plotting area. Geom for easily drawing lines separating the quadrants of a plot.
- Stats for filtering-out/filtering-in observations in regions of a panel or group where the density of observations is high. Statistics simultaneously computing summaries, optionally using different functions, along x and y. Stat computing quadrant counts.
- Position functions implementing multi-directional nudging based on the data.
- Scales. Pseudo-scales supporting npc coordinates for x and y.
- Specializations of the `ggplot()` generic accepting time series objects of classes `ts` and `xts` as data argument.

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()`, `position_nudge_center()`, `position_nudge_line()` and `position_nudge_to()` from this package. This package includes code copied and/or modified from that in package 'ggplot2'.

Author(s)

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

Other contributors:

- Kamil Slowikowski (ORCID) [contributor]
- Michał Krassowski (ORCID) [contributor]
- Daniel Sabanés Bové (ORCID) [contributor]
- Stella Banjo [contributor]

References

Package 'ggplot2' documentation is available at https://ggplot2.tidyverse.org/
Package 'ggplot2' source code at https://github.com/tidyverse/ggplot2

See Also

Useful links:

- https://docs.r4photobiology.info/ggpp/
- https://github.com/aphalo/ggpp
- Report bugs at https://github.com/aphalo/ggpp/issues

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.

Value

A plot layer instance.

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)
```

---

birch.df *Birch seedlings' size*

Description

A dataset containing the measurements on 350 birch seedlings.

Usage

```
birch.df

birch_dw.df
```

Format

A `data.frame` object with 350 rows and 8 variables.

A `data.frame` object with 700 rows and 5 variables.

Details

The data are for seedlings grown in trays with cells or containers of two different volumes. For each of these types of trays, all cells, 1/2 of the cells or 1/4 of the cells contained seedlings. Root-collar diameter (mm), height (cm), dry mass (mg) of stems and roots. Measurements done at the end of the first growing season, after leaf fall.

References

Aphalo, P. J. and Rikala, R. (2003) Field performance of silver-birch planting-stock grown at different spacing and in containers of different volume. *New Forests*, 25:93-108. doi:10.1023/A:1022618810937.

See Also

Other Plant growth and morphology data: `ivy.df`

Examples

```
colnames(birch.df)
head(birch.df)

colnames(birch_dw.df)
head(birch_dw.df)
```

`compute_npcx` *Compute NPC coordinates*

Description

Translate and/or compute NPC (Normalised Parent Coordinates) for use with aesthetics `x` and `y`.

Usage

```
compute_npcx(x, group = 1L, h.step = 0.1, margin.npc = 0.05, each.len = 1)

compute_npcy(y, group = 1L, v.step = 0.1, margin.npc = 0.05, each.len = 1)

as_npcx(x, ...)

as_npcy(y, ...)

compute_npc(a, margin.npc = 0.05)

as_npc(a, margin.npc = 0.05)
```

Arguments

x	numeric or if character, one of "right", "left", "maximum", "minimum", "centre", "center" or "middle".
group	integer vector, ggplot's group id. Used to shift coordinates to avoid overlaps.
h.step, v.step	numeric [0..1] The step size for shifting coordinates in npc units. Usually « 1.
margin.npc	numeric [0..1] The margin added towards the nearest plotting area edge when converting character coordinates into npc. Usually « 1.
each.len	integer The number of steps per group.
y	numeric or if character, one of "top", "bottom", "maximum", "minimum", "centre", "center" or "middle".
...	named arguments passed to compute_npcx() or compute_npcy().
a	numeric or if character, one of "right", "left", "top", "maximum", "minimum", "bottom", "centre", "center" or "middle".

Details

Functions compute_npcx and compute_npcy convert character-encoded positions to npc units and shift positions to avoid overlaps when grouping is active. If numeric, they validate the npc values. Function compute_npcx does the translation either for both x and y aesthetics, but does not implement a shift for grpups. Functions as_npcx(), as_npcy() and as_npc() are wrappers on these functions that return the value as objects of class "AsIs" so that in 'ggplot2' >= 3.5.0 they can be used with any layer function.

These functions use NPC (Normalised Parent Coordinates) instead of data coordinates. They translate named positions into numeric values in [0..1] and they can also shift the position according to the group, e.g., for each increase in the group number displace the position inwards or outwards, by a user-supplied distance. They make it possible to set automatically set default positions for grouped text labels.

Out of bounds numeric values are constrained to [0..1]. Unrecognized character values are silently converted into NA_integer_.

Value

A numeric vector with values in the range [0..1] representing npc coordinates.

Note

The *as_npc()* functions make it easier the use of NPC coordinates with 'ggplot2' >= 3.5.0. The _compute_ functions are used by several layer functions in packages 'ggpp' and 'ggpmisc', are compatible with 'ggplot2' <= 3.4.4 and can be useful to developers of other 'ggplot2' extensions.

Examples

```
compute_npcx("right")
compute_npcx(c("left", "right"))
compute_npcx(c("minimum", "maximum"))
compute_npcx(c("left", "right"), margin.npc = 0)
compute_npcy("bottom")
compute_npcy("bottom", group = 1L:3L)
compute_npcy("bottom", group = 1L:3L, v.step = 0.2)
compute_npcy("bottom", group = 2L)
compute_npcx(0.5)
compute_npcx(1)
compute_npcx(-2)

as_npc("right")
class(as_npc("right"))
class(compute_npcx("right"))
```

---

dark_or_light *Chose between dark and light color*

Description

Chose between a pair of contrasting dark and light colors based on a weighted mean of RGB channels of a color. This function implements a simple approach to the choice for a color of a plot element to ensure it is visible against a background color.

Usage

```
dark_or_light(
  colors,
  threshold = 0.45,
  dark.color = "black",
  light.color = "white"
)
```

Arguments

colors	A vector of color definitions or color names in the background.
threshold	numeric A value of luminance in [0..1] indicating the switch point between dark and light background.

dark.color, light.color
: A color definition or color name to return as dark and light colors to contrast light and dark backgrounds respectively.

Details

The switch between dark and light color is based on a quick and dirty approximation of the luminance of colors computed from RGB values. This easily computed approximation seems to work well enough. The default threshold chosen for a switch between black and white may need to be adjusted for other pairs of colors. Graphic devices can differ in the color spaces they support, but this is unlikely to affect the choice between black and white or other pairs of colors with large differences in luminance.

Note

The current implementation of dark_or_light() ignores alpha, the transparency component, of all its arguments.

Examples

```
dark_or_light("yellow")
dark_or_light("darkblue")
dark_or_light(c("darkblue", "yellow", "red"))
dark_or_light("#FFFFFF")
dark_or_light("#FFFFFF", dark.color = "darkblue", light.color = "lightgrey")
dark_or_light("#000000", dark.color = "darkblue", light.color = "lightgrey")
```

---

geom_grob *Inset graphical objects*

Description

geom_grob and geom_grob_npc add Grobs as insets to the ggplot using syntax similar to that of geom_text, geom_text_s and geom_text_npc. In most respects they behave as any other ggplot geometry: they add a layer containing one or more grobs and grouping and faceting works as usual. The most common use of geom_grob is to add data labels that are graphical objects rather than text. geom_grob_npc is used to add grobs as annotations to plots, but contrary to layer function annotate(), geom_grob_npc is data driven and respects grouping and facets, thus plot insets can differ among panels. Of these two geoms only geom_grob supports the plotting of segments, as geom_grob_npc uses a coordinate system that is unrelated to data units and data.

Usage

```
geom_grob(
  mapping = NULL,
  data = NULL,
  stat = "identity",
```

```
  position = "identity",
  ...,
  nudge_x = 0,
  nudge_y = 0,
  default.colour = NULL,
  default.color = default.colour,
  colour.target = "segment",
  color.target = colour.target,
  default.alpha = 1,
  alpha.target = "segment",
  add.segments = TRUE,
  box.padding = 0.25,
  point.padding = 1e-06,
  segment.linewidth = 0.5,
  min.segment.length = 0,
  arrow = NULL,
  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. 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.
nudge_x, nudge_y	Horizontal and vertical adjustments to nudge the starting position of each "label". The units for nudge_x and nudge_y are the same as for the data units on the x-axis and y-axis.
default.colour, default.color	A colour definition to use for elements not targeted by the colour aesthetic.

`colour.target`, `color.target`
: A vector of character strings; `"all"`, `"text"`, `"box"` and `"segment"` or `"none"`.

`default.alpha`
: numeric in [0..1] A transparency value to use for elements not targeted by the alpha aesthetic.

`alpha.target`
: A vector of character strings; `"all"`, `"text"`, `"segment"`, `"box"`, `"box.line"`, and `"box.fill"` or `"none"`.

`add.segments`
: logical Display connecting segments or arrows between original positions and displaced ones if both are available.

`box.padding`, `point.padding`
: numeric By how much each end of the segments should shortened in mm.

`segment.linewidth`
: numeric Width of the segments or arrows in mm.

`min.segment.length`
: numeric Segments shorter that the minimum length are not rendered, in mm.

`arrow`
: specification for arrow heads, as created by `arrow`

`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

You can modify the size of insets with the `vp.width` and `vp.height` aesthetics. These can take 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 1/5. Thus, in contrast to `geom_text`, `geom_label`, `geom_text_s` and `geom_label_s` the size of the insets remains the same relative to the size of the plotting area irrespective of the size the plot is rendered at. The aspect ratio of insets is preserved and size is adjusted until the whole inset fits within the viewport.

By default `geom_grob` uses `position_nudge_center` and justification `"position"`, while `geom_grob_npc` uses `position_nudge` and justification `"inward"`. In contrast to `position_nudge`, `position_nudge_center` and all other position functions defined in packages 'ggpp' keep the original coordinates thus allowing the plotting of connecting segments and arrows.

`geom_grob` and `geom_grob_npc` expect a list of graphic objects ("grob") to be mapped to the `label` aesthetic. These geoms work with tibbles or data frames as `data` as they both support `list` objects as member variables.

The `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 its *x* and *y* coordinates in the data, and `angle` is used to rotate the grob as a whole.

Value

A plot layer instance.

Plot boundaries and clipping

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 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"`). In addition, you can use special alignments for justification including `"position"`, `"inward"` and `"outward"`. Inward always aligns text towards the center of the plotting area, and outward aligns it away from the center of the plotting area. If tagged with `_mean` or `_median` (e.g., `"outward_mean"`) the mean or median of the data in the panel along the corresponding axis is used as center. If the characters following the underscore represent a number (e.g., `"outward_10.5"`) the reference point will be this value in data units. Position justification is computed based on the direction of the displacement of the position of the label so that each individual text or label is justified outwards from its original position. The default justification is `"position"`.

If no position displacement is applied, or a position function defined in 'ggplot2' is used, these geometries behave similarly to the corresponding ones from package 'ggplot2' with a default justification of `0.5` and no segment drawn.

Position functions

Many layer functions from package 'ggpp' are designed to work seamlessly with position functions that keep, rather than discard, the original `x` and `y` positions in `data` when computing a new displaced position. See `position_nudge_keep`, `position_dodge_keep`, `position_jitter_keep`, `position_nudge_center`, `position_nudge_line`, `position_nudge_to`, `position_dodgenudge`, `position_jitternudge`, and `position_stacknudge` for examples and details of their use.

Note

The insets are stored nested within the main ggplot object and contain their own copy of the data, and are rendered as grid grobs as normal ggplots at the time the main ggplot is rendered. They can have different themes.

Use `annotate` as redefined in 'ggpp' when adding insets as annotations (automatically available unless 'ggpp' is not attached). `annotate` cannot be used with the `npcx` and `npcy` pseudo-aesthetics.

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

`grid-package`, `geom_text`, and other documentation of package 'ggplot2'.

Examples

```
library(tibble)
df <- tibble(x = 2, y = 15, grob = list(grid::circleGrob(r = 0.2)))

# without nudging no segments are drawn
ggplot(data = mtcars,
       aes(wt, mpg)) +
  geom_point(aes(colour = factor(cyl))) +
  geom_grob(data = df,
            aes(x, y, label = grob))

# with nudging segments are drawn
ggplot(data = mtcars,
       aes(wt, mpg)) +
  geom_point(aes(colour = factor(cyl))) +
  geom_grob(data = df,
            aes(x, y, label = grob),
            nudge_x = 0.5,
            colour = "red",
            hjust = 0.5,
            vjust = 0.5)

ggplot(data = mtcars,
       aes(wt, mpg)) +
  geom_point(aes(colour = factor(cyl))) +
  geom_grob(data = df,
            aes(x, y, label = grob),
            nudge_x = 0.5,
            colour = "red",
            colour.target = "none",
            hjust = 0.5,
            vjust = 0.5)

# with nudging plotting of segments can be disabled
ggplot(data = mtcars,
       aes(wt, mpg)) +
  geom_point(aes(colour = factor(cyl))) +
  geom_grob(data = df,
            aes(x, y, label = grob),
            add.segments = FALSE,
            nudge_x = 0.5,
            hjust = 0.5,
            vjust = 0.5)
```

---

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, using pseudo-aesthetics. Their intended use is to add annotations to a plot.

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,
  size.unit = "mm",
  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. 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.
size.unit	How the 'size' aesthetic is interpreted: as millimetres ('"mm"', default), points ('"pt"'), centimetres ('"cm"'), inches ('"in"'), or picas ('"pc"').
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

These geoms are identical to 'ggplot2' geom_text and geom_label except that they interpret npcx and npcy positions in npc units. They translate npcx and npcy coordinates using a pseudo-aesthetic with a fixed scale, the translation is done separately for each plot panel. All aesthetics other than *x* and *y* and grouping work as in normal geoms. These include linetype and angle in geom_label_npc().

Alignment

With textual positions and groups a shift is added to successive labels to avoid overlaps. The shift is based on grouping, however unused levels are not dropped. In plots with faceting, if not all groups appear in each panel, there will be blank spaces in between labels. To solve this pass numeric values for the npc coordinates of each label instead of character strings.

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"). In addition, you can use special alignments for justification including "position", "inward" and "outward". Inward always aligns text towards the center of the plotting area, and outward aligns it away from the center of the plotting area. If tagged with _mean or _median (e.g., "outward_mean") the mean or median of the data in the panel along the corresponding axis is used as center. If the characters following the underscore represent a number (e.g., "outward_10.5") the reference point will be this value in data units. Position justification is computed based on the direction of the displacement of the position of the label so that each individual text or label is justified outwards from its original position. The default justification is "position".

If no position displacement is applied, or a position function defined in 'ggplot2' is used, these geometries behave similarly to the corresponding ones from package 'ggplot2' with a default justification of `0.5` and no segment drawn.

Plot boundaries and clipping

Note that when you change the scale limits for *x* and/or *y* of a plot, text labels stay the same size, as determined by the `size` aesthetic, given in millimetres. The actual size as seen in the plotted output is decided during the rendering of the plot to a graphics device. Limits are expanded only to include the anchor point of the labels because the "width" and "height" of a text element are 0 (as seen by ggplot2). Text labels do have height and width, but in grid units, not data units.

See Also

`geom_text` and `geom_label` for additional details.

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(df) +
  geom_text_npc(aes(npcx = x.chr, npcy = y.chr, label = text),
                angle = 90)

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))

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

```
                angle = 90) # ignored by ggplot2 < 3.5.0
```

`geom_label_pairwise` *Label pairwise comparisons*

Description

Add a plot layer with a text label and a segment connecting two values along the `x` aesthetic. These are usually two levels of a factor mapped to the `x` aesthetic when used to report significance or highlighting pairwise comparisons.

Usage

```
geom_label_pairwise(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  parse = FALSE,
  nudge_x = 0,
  nudge_y = 0,
  default.colour = NULL,
  default.color = default.colour,
  colour.target = "all",
  color.target = colour.target,
  default.alpha = NA,
  alpha.target = "segment",
  label.padding = grid::unit(0.25, "lines"),
  label.r = grid::unit(0.15, "lines"),
  segment.linewidth = 0.5,
  arrow = NULL,
  size.unit = "mm",
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = FALSE
)

geom_text_pairwise(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  parse = FALSE,
  nudge_x = 0,
```

```
  nudge_y = 0,
  default.colour = "black",
  default.color = default.colour,
  colour.target = "all",
  color.target = colour.target,
  default.alpha = NA,
  alpha.target = "all",
  segment.linewidth = 0.5,
  arrow = NULL,
  check_overlap = FALSE,
  size.unit = "mm",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = FALSE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes. With inherit.aes = FALSE (the default) it is not combined with the default mapping at the top level of the plot. You always need to supply a mapping unless you set inherit.aes = TRUE.
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.
default.colour, default.color	A colour definition to use for elements not targeted by the colour aesthetic.
colour.target, color.target	A vector of character strings; "all", "text", "segment", "box", "box.line", and "box.fill" or "none".
default.alpha	numeric in [0..1] A transparency value to use for elements not targeted by the alpha aesthetic.

alpha.target	A vector of character strings; "all", "text", "segment", "box", "box.line", and "box.fill" or "none".
label.padding	Amount of padding around label. Defaults to 0.25 lines.
label.r	Radius of rounded corners. Defaults to 0.15 lines.
segment.linewidth	numeric Width of the segments or arrows in mm.
arrow	specification for arrow heads, as created by arrow
size.unit	How the 'size' aesthetic is interpreted: as millimetres ('"mm"', default), points ('"pt"'), centimetres ('"cm"'), inches ('"in"'), or picas ('"pc"').
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 includes a legend if any aesthetics are mapped. FALSE, the default, never includes it, and TRUE always includes it.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining them.
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.

Details

Geometries geom_text_pairwise() and geom_label_pairwise() have an interface similar to that of geom_text and geom_label, but add a segment connecting two values along x. In the most frequent use case they add a segment connecting pairs of levels from a grouping factor mapped to the *x* or *y* aesthetic. They can also be used to label ranges of values.

The segment extends from xmin to xmax, and the text label is located at x with a default that positions the label at the centre of the bar. The ends of the bar can be terminated with arrow heads given by parameter arrow, with a default of a plain segment without arrow tips. The text label is located slightly above the segment by the default value of vjust in geom_text_pairwise() and on top of the segment in geom_label_pairwise().

Layer functions geom_text_pairwise() and geom_label_pairwise() use by default position_nudge. Nudging affects both text label and bar, and its default of no displacement will very rarely need to be changed.

Differently to geom_text_repel() and geom_label_repel(), geom_text_pairwise() and geom_label_pairwise() do not make use of additional aesthetics for the segments or boxes, but instead allow the choice of which elements are targeted by the usual 'ggplot2' aesthetics and which are rendered using a default constant value. In the grammar of graphics using the same aesthetic with multiple meanings is not allowed, thus, the approach used in package 'ggpp' attempts to enforce this.

Value

A plot layer instance.

Under development!

This geometry is still under development and its user interface subject to change.

Plot boundaries and clipping

Note that when you change the scale limits for *x* and/or *y* of a plot, text labels stay the same size, as determined by the `size` aesthetic, given in millimetres. The actual size as seen in the plotted output is decided during the rendering of the plot to a graphics device. Limits are expanded only to include the anchor point of the labels because the "width" and "height" of a text element are 0 (as seen by ggplot2). Text labels do have height and width, but in grid units, not data units. Either function `expand_limits` or the scale expansion can be used to ensure text labels remain within the plotting area.

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"`). Values outside the range 0..1 displace the text label so that the anchor point is outside the text label. In addition, you can use special alignments for justification including `"position"`, `"inward"` and `"outward"`. Inward always aligns text towards the center of the plotting area, and outward aligns it away from the center of the plotting area. If tagged with `_mean` or `_median` (e.g., `"outward_mean"`) the mean or median of the data in the panel along the corresponding axis is used as center. If the characters following the underscore represent a number (e.g., `"outward_10.5"`) the reference point will be this value in data units. Position justification is computed based on the direction of the displacement of the position of the label so that each individual text or label is justified outwards from its original position. The default justification is `"identity"`.

Aesthetics

Layer functions `geom_text_pairwise()` and `geom_label_pairwise()` require aesthetics `xmin`, `xmax`, `x`, `y` and `label` and support aesthetics: `alpha`, `colour`, `group`, `size` (of text), `family`, `fontface`, `linewidth`, `linetype`, `hjust` and `vjust`. In addition, `geom_text_pairwise` supports `angle` and `geom_label_pairwise` supports `fill`. See `aes_colour_fill_alpha`, `aes_linetype_size_shape`, `aes_position`, and `aes_group_order`.

In 'ggplot2' `linewidth` when applied to the border of the box drawn by `geom_label()` is given in points rather than in mm because of a historical error in the code. In other geometries such as `geom_segment()` `linewidth` is given in mm. As in `geom_label_pairwise()` it is important to remain consistent among different `linewidth` specifications, mm are used both for the box border and linking segment. To imitate the behaviour of `geom_label()` a correction factor of 0.75 (more exactly 1 pt = 0.7528 mm) can be used for the line width of the border of the box.

See Also

`geom_text_s`, `geom_label_s`, `geom_text`, `geom_label` and other documentation of package 'ggplot2'.

Examples

```
my.cars <- mtcars
my.cars$name <- rownames(my.cars)
p1 <- ggplot(my.cars, aes(factor(cyl), mpg)) +
       geom_boxplot(width = 0.33)
```

```
# With a factor mapped to x, highlight pairs

my.pairs <-
  data.frame(A = 1:2, B = 2:3, bar.height = c(12, 30),
             p.value = c(0.01, 0.05678))
p1 +
  geom_text_pairwise(data = my.pairs,
                     aes(xmin = A, xmax = B,
                         y = bar.height,
                         label = p.value),
                     parse = TRUE)

p1 +
  geom_text_pairwise(data = my.pairs,
                     aes(xmin = A, xmax = B,
                         y = bar.height,
                         label = sprintf("italic(P)~`=`~%.2f", p.value)),
                     arrow = grid::arrow(angle = 90,
                                         length = unit(1, "mm"),
                                         ends = "both"),
                     parse = TRUE)

p1 +
  geom_text_pairwise(data = my.pairs,
                     aes(xmin = A, xmax = B,
                         y = bar.height,
                         label = sprintf("italic(P)~`=`~%.2f", p.value)),
                     colour = "red",
                     arrow = grid::arrow(angle = 90,
                                         length = unit(1, "mm"),
                                         ends = "both"),
                     parse = TRUE)

p1 +
  geom_label_pairwise(data = my.pairs,
                      aes(xmin = A, xmax = B,
                          y = bar.height,
                          label = sprintf("italic(P)~`=`~%.2f", p.value)),
                      colour = "red", size = 2.75,
                      arrow = grid::arrow(angle = 30,
                                          length = unit(1.5, "mm"),
                                          ends = "both"),
                      parse = TRUE)

p1 +
  geom_text_pairwise(data = my.pairs,
                     aes(xmin = A, xmax = B,
                         y = bar.height,
                         label = sprintf("italic(P)~`=`~%.2f", p.value)),
                     colour = "red", colour.target = "segment",
                     arrow = grid::arrow(angle = 90,
                                         length = unit(1, "mm"),
                                         ends = "both"),
```

```
                     parse = TRUE)

p1 +
  geom_text_pairwise(data = my.pairs,
                     aes(xmin = A, xmax = B,
                         y = bar.height,
                         label = sprintf("italic(P)~`=`~%.2f", p.value)),
                     colour = "red", colour.target = "text",
                     arrow = grid::arrow(angle = 90,
                                         length = unit(1, "mm"),
                                         ends = "both"),
                     parse = TRUE)

# with a numeric vector mapped to x, indicate range

p2 <-
  ggplot(my.cars, aes(disp, mpg)) +
    geom_point()

my.ranges <-
  data.frame(A = c(50, 400),
             B = c(200, 500),
             bar.height = 5,
             text = c("small", "large"))

p2 +
  geom_text_pairwise(data = my.ranges,
                     aes(xmin = A, xmax = B,
                     y = bar.height, label = text))

p2 +
  geom_text_pairwise(data = my.ranges,
                     aes(xmin = A, xmax = B,
                         y = bar.height, label = text),
                     angle = 90, hjust = -0.1)

p2 +
  geom_label_pairwise(data = my.ranges,
                      aes(xmin = A, xmax = B,
                          y = bar.height, label = text),
                     angle = 90, hjust = -0.1)

p2 +
  geom_label_pairwise(data = my.ranges,
                      aes(xmin = A, xmax = B,
                          y = bar.height, label = text))

p2 +
  geom_text_pairwise(data = my.ranges,
                     aes(xmin = A, xmax = B,
                         y = bar.height, label = text),
                     arrow = grid::arrow(ends = "both", length = unit(2, "mm")))
```

geom_label_s *Linked Text*

Description

Linked text geometries are most useful for adding data labels to plots. ‘geom_text_s()‘ and ‘geom_label_s()‘ add text to the plot and for nudged positions link the original location to the nudged text with a segment or arrow.

Usage

```
geom_label_s(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  parse = FALSE,
  nudge_x = 0,
  nudge_y = 0,
  default.colour = NULL,
  default.color = default.colour,
  colour.target = c("text", "box"),
  color.target = colour.target,
  default.alpha = NA,
  alpha.target = "all",
  label.padding = grid::unit(0.25, "lines"),
  label.r = grid::unit(0.15, "lines"),
  segment.linewidth = 0.5,
  add.segments = TRUE,
  box.padding = 1e-06,
  point.padding = 1e-06,
  min.segment.length = 0,
  arrow = NULL,
  size.unit = "mm",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_text_s(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  parse = FALSE,
```

```
  nudge_x = 0,
  nudge_y = 0,
  default.colour = NULL,
  default.color = default.colour,
  colour.target = "text",
  color.target = colour.target,
  default.alpha = NA,
  alpha.target = "all",
  add.segments = TRUE,
  box.padding = 0.25,
  point.padding = 1e-06,
  segment.linewidth = 0.5,
  min.segment.length = 0,
  arrow = NULL,
  check_overlap = FALSE,
  size.unit = "mm",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes. If specified and with inherit.aes = TRUE (the default), it 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.
default.colour, default.color	A colour definition to use for elements not targeted by the colour aesthetic.

colour.target, color.target
: A vector of character strings; "all", "text", "segment", "box", "box.line", and "box.fill" or "none".

default.alpha
: numeric in [0..1] A transparency value to use for elements not targeted by the alpha aesthetic.

alpha.target
: A vector of character strings; "all", "text", "segment", "box", "box.line", and "box.fill" or "none".

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

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

segment.linewidth
: numeric Width of the segments or arrows in mm.

add.segments
: logical Display connecting segments or arrows between original positions and displaced ones if both are available.

box.padding, point.padding
: numeric By how much each end of the segments should shortened in mm.

min.segment.length
: numeric Segments shorter that the minimum length are not rendered, in mm.

arrow
: specification for arrow heads, as created by arrow

size.unit
: How the 'size' aesthetic is interpreted: as millimetres ('"mm"', default), points ('"pt"'), centimetres ('"cm"'), inches ('"in"'), or picas ('"pc"').

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 a legend if any aesthetics are mapped. FALSE never includes it, and TRUE always includes it.

inherit.aes
: If FALSE, overrides the default aesthetics, rather than combining 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. 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.

Details

Geometries geom_text_s() and geom_label_s() have an interface similar to that of geom_text and geom_label, but support additional features. Similarly to geom_text_repel() and geom_label_repel() when used together with position functions defined in package 'ggpp' they draw a segment linking the label at a displaced position to the original position, usually a point corresponding to an observation to which the label refers. Another difference is that they allow control of to which graphical elements the mappings to colour and alpha aesthetics are applied. Differently to geom_label(), geom_label_s() obeys aesthetic mappings to linewidth and linetype applied to the line at the edge of the label box. These features are reflected in the plot key, except for the segment, assumed not to be used to display information only in coordination with other graphic elements.

In geom_label_s() the default fill is similar to "white" but with its alpha component set to 0.75. This differs from "white" used in geom_label(): the default fill is semitransparent with

the intention that accidental occlusion of observations is obvious irrespective of the order in which layers are added to the plot.

Layer functions `geom_text_s()` and `geom_label_s()` use by default `position_nudge_keep` which is backwards compatible with `position_nudge`. In contrast to `position_nudge`, `position_nudge_keep` and all other position functions defined in packages 'ggpp' and 'ggrepel' keep the original coordinates, thus allowing the plotting of connecting segments and arrows.

Differently to `geom_text_repel()` and `geom_label_repel()`, `geom_text_s()` and `geom_label_s()` do not make use of additional aesthetics for the segments or boxes, but instead allow the choice of which elements are targeted by the aesthetics and which are rendered in a default colour. In the grammar of graphics using the same aesthetic with multiple meanings is not allowed, thus, the approach used in the geometry layer functions from package 'ggpp' attempts to enforce this.

Value

A plot layer instance.

Plot boundaries and clipping

Note that when you change the scale limits for *x* and/or *y* of a plot, text labels stay the same size, as determined by the `size` aesthetic, given in millimetres. The actual size as seen in the plotted output is decided during the rendering of the plot to a graphics device. Limits are expanded only to include the anchor point of the labels because the "width" and "height" of a text element are 0 (as seen by ggplot2). Text labels do have height and width, but in grid units, not data units.

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"`). In addition, you can use special alignments for justification including `"position"`, `"inward"` and `"outward"`. Inward always aligns text towards the center of the plotting area, and outward aligns it away from the center of the plotting area. If tagged with `_mean` or `_median` (e.g., `"outward_mean"`) the mean or median of the data in the panel along the corresponding axis is used as center. If the characters following the underscore represent a number (e.g., `"outward_10.5"`) the reference point will be this value in data units. Position justification is computed based on the direction of the displacement of the position of the label so that each individual text or label is justified outwards from its original position. The default justification is `"position"`.

If no position displacement is applied, or a position function defined in 'ggplot2' is used, these geometries behave similarly to the corresponding ones from package 'ggplot2' with a default justification of `0.5` and no segment drawn.

Differences from earlier versions

The user interface is for the most part stable starting from 'ggpp' (== 0.5.7). In 'ggpp' (== 0.5.0) support for aesthetics related to segments was removed, and replaced by parameters and a new mechanism for targeting the usual `colour` and `alpha` aesthetics to text, border, and segment.

Aesthetics

Layer functions `geom_text_s()` and `geom_label_s()` require aesthetics `x`, `y` and `label` and support aesthetics: `alpha`, `colour`, `group`, `size` (of text), `family`, `fontface`, `lineheight`, `hjust` and `vjust`. In addition, `geom_text_s` supports `angle` and `geom_label_s` supports `fill`, `linewidth` and `linetype`. See `aes_colour_fill_alpha`, `aes_linetype_size_shape`, `aes_position`, and `aes_group_order`.

In 'ggplot2' `linewidth` when applied to the border of the box drawn by `geom_label()` is given in points rather than in mm because of a historical error in the code. In other geometries such as `geom_segment()` `linewidth` is given in mm. As in `geom_label_s()` it is important to remain consistent among different `linewidth` specifications, mm are used both for the box border and linking segment. To imitate the behaviour of `geom_label()` a correction factor of 0.75 (more exactly 1 pt = 0.7528 mm) can be used for the line width of the border of the box.

Position functions

Many layer functions from package 'ggpp' are designed to work seamlessly with position functions that keep, rather than discard, the original `x` and `y` positions in `data` when computing a new displaced position. See `position_nudge_keep`, `position_dodge_keep`, `position_jitter_keep`, `position_nudge_center`, `position_nudge_line`, `position_nudge_to`, `position_dodgenudge`, `position_jitternudge`, and `position_stacknudge` for examples and details of their use.

See Also

`geom_text`, `geom_label` and other documentation of package 'ggplot2'.

Examples

```
my.cars <- mtcars[c(TRUE, FALSE, FALSE, FALSE), ]
my.cars$name <- rownames(my.cars)

# no nudging
ggplot(my.cars, aes(wt, mpg, label = name)) +
  geom_text_s() +
  expand_limits(x = c(2, 6))

# base plot
p <- ggplot(my.cars, aes(wt, mpg, label = name)) +
       geom_point()

# Using nudging
p +
  geom_text_s(nudge_x = 0.12) +
  expand_limits(x = 6.2)
p +
  geom_text_s(nudge_x = -0.12) +
  expand_limits(x = 1.5)
p +
  geom_text_s(nudge_x = 0.12,
              arrow = arrow(length = grid::unit(1.5, "mm")),
              point.padding = 0.4) +
```

```
  expand_limits(x = 6.2)
p +
  geom_text_s(nudge_y = 0.1, nudge_x = 0.07) +
  expand_limits(x = 6.2)
p +
  geom_text_s(nudge_y = 1, angle = 90) +
  expand_limits(y = 30)
p +
  geom_text_s(angle = 90, nudge_y = 1,
              arrow = arrow(length = grid::unit(1.5, "mm")),
              colour.target = "segment", colour = "red") +
  expand_limits(y = 30)
p +
  geom_text_s(aes(colour = factor(cyl)),
              angle = 90, nudge_y = 1,
              arrow = arrow(length = grid::unit(1.5, "mm")),
              alpha.target = "segment", alpha = 0.3) +
  expand_limits(y = 30)

p +
  geom_label_s(nudge_x = 0.12) +
  expand_limits(x = 6.2)
p +
  geom_label_s(nudge_x = 0.12, linetype = "dotted", linewidth = 0.3) +
  expand_limits(x = 6.2)
p +
  geom_label_s(aes(colour = factor(cyl)),
               nudge_x = 0.12,
               colour.target = "box",
               linewidth = 0.5,
               label.r = unit(0, "lines")) +
  expand_limits(x = 6.2)
p +
  geom_label_s(nudge_x = 0.12, linewidth = 0) +
  expand_limits(x = 6.2)

# No segments
p +
  geom_label_s(nudge_x = 0.05, segment.linewidth = 0) +
  expand_limits(x = 6.2)

# Nudging away from arbitrary point
p +
  geom_label_s(hjust = "outward_1", nudge_x = 0.12) +
  expand_limits(x = 6.2)
p +
  geom_label_s(hjust = "inward_3", nudge_y = 0.4)

p +
  geom_label_s(nudge_y = 1, angle = 90) +
  expand_limits(y = 30)

# Add aesthetic mappings and adjust arrows
```

```
p +
  geom_text_s(aes(colour = factor(cyl)),
              angle = 90,
              nudge_y = 1,
              arrow = arrow(angle = 20,
                            length = grid::unit(1.5, "mm"),
                            ends = "first",
                            type = "closed")) +
  scale_colour_discrete(l = 40) + # luminance, make colours darker
  expand_limits(y = 27)

p +
  geom_text_s(aes(colour = factor(cyl)),
              angle = 90,
              nudge_y = 1,
              arrow = arrow(angle = 20,
                            length = grid::unit(1.5, "mm"),
                            ends = "first",
                            type = "closed")) +
  scale_colour_discrete(l = 40) + # luminance, make colours darker
  expand_limits(y = 27)

p +
  geom_label_s(aes(colour = factor(cyl)),
              colour.target = c("box", "text"),
              nudge_x = 0.3,
              arrow = arrow(angle = 20,
                            length = grid::unit(1/3, "lines"))) +
  scale_colour_discrete(l = 40) + # luminance, make colours darker
  expand_limits(x = 7)

p +
  geom_label_s(aes(colour = factor(cyl)),
              nudge_x = 0.3,
              colour.target = c("box", "segment"),
              linewidth = 0.5,
              arrow = arrow(angle = 20,
                            length = grid::unit(1/3, "lines"))) +
  scale_colour_discrete(l = 40) + # luminance, make colours darker
  expand_limits(x = 7)

p +
  geom_label_s(aes(colour = factor(cyl), fill = factor(cyl)),
              nudge_x = 0.3,
              alpha.target = "box",
              alpha = 0.1,
              linewidth = 0.5,
              arrow = arrow(angle = 20,
                            length = grid::unit(1/3, "lines"))) +
  scale_colour_discrete(l = 40) + # luminance, make colours darker
  expand_limits(x = 7)#' # Scale height of text, rather than sqrt(height)

p +
```

```
  geom_text_s(aes(size = wt), nudge_x = -0.1) +
  scale_radius(range = c(3,6)) + # override scale_area()
    expand_limits(x = c(1.8, 5.5))
```

---

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 and geom_text_s. In most respects they behave as any other ggplot geometry: they add a layer containing one or more grobs and grouping and faceting works as usual. The most common use of geom_plot is to add data labels that are themselves ggplots rather than text. geom_plot_npc is used to add ggplots as annotations to plots, but contrary to layer function annotate(), geom_plot_npc is data driven and respects grouping and facets, thus plot insets can differ among panels.

Usage

```
geom_plot(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  nudge_x = 0,
  nudge_y = 0,
  default.colour = NULL,
  default.color = default.colour,
  colour.target = "box",
  color.target = colour.target,
  default.alpha = 1,
  alpha.target = "all",
  add.segments = TRUE,
  box.padding = 0.25,
  point.padding = 1e-06,
  segment.linewidth = 0.5,
  min.segment.length = 0,
  arrow = NULL,
  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. 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.
nudge_x, nudge_y	Horizontal and vertical adjustments to nudge the starting position of each "label". The units for nudge_x and nudge_y are the same as for the data units on the x-axis and y-axis.
default.colour, default.color	A colour definition to use for elements not targeted by the colour aesthetic.
colour.target, color.target	A vector of character strings; "all", "text", "box" and "segment".
default.alpha	numeric in [0..1] A transparency value to use for elements not targeted by the alpha aesthetic.
alpha.target	A vector of character strings; "all", "text", "segment", "box", "box.line", and "box.fill".
add.segments	logical Display connecting segments or arrows between original positions and displaced ones if both are available.
box.padding, point.padding	numeric By how much each end of the segments should shortened in mm.
segment.linewidth	numeric Width of the segments or arrows in mm.
min.segment.length	numeric Segments shorter that the minimum length are not rendered, in mm.
arrow	specification for arrow heads, as created by arrow
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

You can modify the size of inset plots with the `vp.width` and `vp.height` aesthetics. These can take 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 1/5. Thus, in contrast to `geom_text` and `geom_text_s` the size of the insets remains the same relative to the size of the plotting area irrespective of how the plot is rendered. The aspect ratio of insets is preserved and size is adjusted until the whole inset fits within the viewport.

`geom_plot` and `geom_plot_npc` expect a list of ggplot objects (`"gg"` class) to be mapped to the `label` aesthetic. These geoms work with tibbles or data frames as `data` as they both support `list` objects as member variables.

The `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 its x and y coordinates in the data, and `angle` is used to rotate the plot as a whole.

Of these two geoms only `geom_plot` supports the plotting of segments, as `geom_plot_npc` uses a coordinate system that is unrelated to data units and data.In the case of `geom_plot_npc()`, `npcx` and `npcy` aesthetics determine the position of the inset plot.

Value

A plot layer instance.

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"`). In addition, you can use special alignments for justification including `"position"`, `"inward"` and `"outward"`. Inward always aligns text towards the center of the plotting area, and outward aligns it away from the center of the plotting area. If tagged with `_mean` or `_median` (e.g., `"outward_mean"`) the mean or median of the data in the panel along the corresponding axis is used as center. If the characters following the underscore represent a number (e.g., `"outward_10.5"`) the reference point will be this value in data units. Position justification is computed based on the direction of the displacement of the position of the label so that each individual text or label is justified outwards from its original position. The default justification is `"position"`.

If no position displacement is applied, or a position function defined in 'ggplot2' is used, these geometries behave similarly to the corresponding ones from package 'ggplot2' with a default justification of `0.5` and no segment drawn.

Position functions

Many layer functions from package 'ggpp' are designed to work seamlessly with position functions that keep, rather than discard, the original `x` and `y` positions in `data` when computing a new displaced position. See `position_nudge_keep`, `position_dodge_keep`, `position_jitter_keep`, `position_nudge_center`, `position_nudge_line`, `position_nudge_to`, `position_dodgenudge`, `position_jitternudge`, and `position_stacknudge` for examples and details of their use.

Plot boundaries and clipping

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.

Note

The insets are stored nested within the main ggplot object and contain their own copy of the data, and are rendered as grid grobs as normal ggplots at the time the main ggplot is rendered. They can have different themes.

Use `annotate` as redefined in 'ggpp' when adding insets as annotations (automatically available unless 'ggpp' is not attached). `annotate` cannot be used with the `npcx` and `npcy` pseudo-aesthetics.

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_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()

p.crop <- p +
          coord_cartesian(xlim = c(3, 4),
          ylim = c(13, 16)) +
          labs(x = NULL, y = NULL) +
          theme_bw(10)

df <- data.frame(x = 0.01,
                 y = 0.01,
                 plot = I(list(p.crop)))
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,
                aes(npcx = x, npcy = y, label = plot,
                vp.width = 1/2, vp.height = 1/4))
```

```
p +
  expand_limits(x = 0, y = 0) +
  geom_plot_npc(data = df,
                aes(npcx = x, npcy = y, label = plot),
                vp.width = 1/4, vp.height = 1/4)

p +
  geom_plot(data = df,
            aes(x = x + 3, y = y + 20, label = plot),
            nudge_x = -1, nudge_y = - 7,
            hjust = 0.5, vjust = 0.5,
            arrow = arrow(length = unit(0.5, "lines")),
            colour = "red",
            vp.width = 1/5, vp.height = 1/5)
```

`geom_point_s` *Points linked by a segment*

Description

The geometry "geom_point_s" provides a super set of the capabilities of geom geom_point from package 'ggplot2' by allowing plotting of arrows or segments joining the original position of displaced observations to their current position rendered as points or graphic symbols. The most common use is to demonstrate the action of different position functions. It can be also used to highlight observations.

Usage

```
geom_point_s(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  nudge_x = 0,
  nudge_y = 0,
  move.point = TRUE,
  arrow = grid::arrow(length = unit(1/3, "lines"), ends = "first"),
  default.colour = NULL,
  default.color = default.colour,
  colour.target = "point",
  color.target = colour.target,
  default.alpha = NA,
  alpha.target = "all",
  add.segments = TRUE,
  box.padding = 0.25,
```

```
  point.padding = 1e-06,
  segment.linewidth = 0.5,
  min.segment.length = 0,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by 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.
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.
move.point	logical If TRUE the point is drawn at the nudged position while if FALSE the point is drawn at the original position.
arrow	specification for arrow heads, as created by arrow
default.colour, default.color	A colour definition to use for elements not targeted by the colour aesthetic.
colour.target, color.target	A character string, one of "all", "point" and "segment" or "none".
default.alpha	numeric in [0..1] A transparency value to use for elements not targeted by the alpha aesthetic.
alpha.target	A character string, one of "all", "segment", "point", or "none".
add.segments	logical Display connecting segments or arrows between original positions and displaced ones if both are available.
box.padding, point.padding	numeric By how much each end of the segments should shortened in mm.
segment.linewidth	numeric Width of the segments or arrows in mm.

`min.segment.length`
: numeric Segments shorter that the minimum length are not rendered, 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`.

Details

The plotting of segments is similar in idea to that implemented in `geom_text_repel` and relies on position functions that rename instead of only replacing the original `x` and `y` coordinates from the `data` object.

By default this geom uses `position_nudge_center` which is backwards compatible with `position_nudge` but provides additional control on the direction of the nudging.

Value

A plot layer instance.

Position functions

Many layer functions from package 'ggpp' are designed to work seamlessly with position functions that keep, rather than discard, the original `x` and `y` positions in `data` when computing a new displaced position. See `position_nudge_keep`, `position_dodge_keep`, `position_jitter_keep`, `position_nudge_center`, `position_nudge_line`, `position_nudge_to`, `position_dodgenudge`, `position_jitternudge`, and `position_stacknudge` for examples and details of their use.

Note

The insets are stored nested within the main ggplot object and contain their own copy of the data, and are rendered as grid grobs as normal ggplots at the time the main ggplot is rendered. They can have different themes.

Use `annotate` as redefined in 'ggpp' when adding insets as annotations (automatically available unless 'ggpp' is not attached). `annotate` cannot be used with the `npcx` and `npcy` pseudo-aesthetics.

See Also

`geom_point`.

Examples

```
# Same output as with geom_point()
ggplot(mpg[1:20, ],
       aes(cyl, hwy)) +
  geom_point_s(colour = "blue")
```

```
# with segment drawn after nudging
ggplot(mpg[1:20, ],
       aes(cyl, hwy, label = drv)) +
  geom_point_s(position = position_nudge_keep(x = 0.2),
               colour = "red") +
  geom_point_s(colour = "blue") +
  expand_limits(x = c(3.5, 8.5))

ggplot(mpg[1:20, ],
       aes(cyl, hwy, label = drv)) +
  geom_point_s(position = position_nudge_keep(x = 0.2),
               colour = "blue",
               move.point = FALSE) +
  expand_limits(x = c(3.5, 8.5))

# with segment drawn after nudging
ggplot(mpg[1:20, ],
       aes(cyl, hwy, label = drv)) +
  geom_point_s(position = position_nudge_keep(x = 0.2),
               colour = "red",
               colour.target = "all") +
  geom_point_s(colour = "blue")

ggplot(mpg[1:20, ],
       aes(cyl, hwy, label = drv)) +
  geom_point_s(position = position_nudge_keep(x = 0.2),
               colour = "red",
               colour.target = "segment") +
  geom_point_s(colour = "blue")

ggplot(mpg[1:20, ],
       aes(cyl, hwy, label = drv)) +
  geom_point_s(position = position_nudge_keep(x = 0.2),
               colour = "red",
               colour.target = "point") +
  geom_point_s(colour = "blue")

ggplot(mpg[1:50, ],
       aes(cyl, hwy, label = drv)) +
  geom_point_s(position = position_jitternudge(width = 0.66, height = 2,
                                               seed = 456,
                                               nudge.from = "jittered",
                                               kept.origin = "original"),
               colour = "red",
               alpha = 0.3, alpha.target = "segment",
               arrow = grid::arrow(length = grid::unit(0.4, "lines"),
                                   ends = "first")) +
  geom_point_s(colour = "blue")
```

---

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 = c("none", "x", "y", "xy"),
  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. 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", "y", or "xy" indicating whether to plot or not lines separating 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 accepts 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()`.

Value

A plot layer instance.

See Also

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

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

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()

ggplot(my.data, aes(x, y)) +
  geom_vhlines(xintercept = c(25, 50, 75),
               yintercept = c(10, 8),
               linetype = "dotted",
               colour = "red") +
  geom_point() +
  theme_bw()
```

`geom_table` *Inset tables*

Description

`geom_table` and `geom_table_npc` add data frames as table insets to the base ggplot, using syntax similar to that of `geom_text` and `geom_text_s`. In most respects they behave as any other ggplot geometry: they add a layer containing one or more grobs and grouping and faceting works as usual. The most common use of `geom_table` is to add data labels that are whole tables rather than text. `geom_table_npc` is used to add tables as annotations to plots, but contrary to layer function `annotate`, `geom_table_npc` is data driven and respects grouping and facets, thus plot insets can differ among panels.

Usage

```
geom_table(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  nudge_x = 0,
  nudge_y = 0,
  default.colour = NA,
```

```
  default.color = default.colour,
  colour.target = "table.text",
  color.target = colour.target,
  default.alpha = 1,
  alpha.target = "all",
  fontsize.scaling = 0.825,
  add.segments = TRUE,
  box.padding = 0.25,
  point.padding = 1e-06,
  segment.linewidth = 0.5,
  min.segment.length = 0,
  arrow = NULL,
  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",
  ...,
  default.colour = NA,
  default.color = default.colour,
  colour.target = "table.text",
  color.target = colour.target,
  default.alpha = 1,
  alpha.target = "all",
  fontsize.scaling = 0.825,
  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. 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.

`nudge_x, nudge_y`
Horizontal and vertical adjustments to nudge the starting position of each "label". The units for `nudge_x` and `nudge_y` are the same as for the data units on the x-axis and y-axis.

`default.colour, default.color`
A colour definition to use for elements not targeted by the colour aesthetic. If `NA` the colours in the table theme are not modified.

`colour.target, color.target`
A vector of character strings with one or more of `"all"`, `"table"`, `"table.text"`, `"table.rules"`, `"segment"` or `"none"`.

`default.alpha` numeric in [0..1] A transparency value to use for elements not targeted by the alpha aesthetic. If `NA` the alpha channel of the colour definitions is not modified.

`alpha.target` A vector of character strings with one or more of `"all"`, `"table"`, `"table.text"`, `"table.rules"` and `"table.canvas"`, `"segment"` or `"none"`.

`fontsize.scaling`
A scaling factor to apply to the *size* aesthetic retrieved from the theme or mapped, applied to table text.

`add.segments` logical Display connecting segments or arrows between original positions and displaced ones if both are available.

`box.padding, point.padding`
numeric By how much each end of the segments should shortened in mm.

`segment.linewidth`
numeric Width of the segments or arrows in mm.

`min.segment.length`
numeric Segments shorter that the minimum length are not rendered, in mm.

`arrow` specification for arrow heads, as created by `arrow`

`table.theme` NULL, list or function A gridExtra ttheme defintion, or a constructor for a ttheme or NULL for default. If `nULL` the theme is retrieved from R option at the time the plot is rendered.

`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

`geom_table` and `geom_table_npc` expect a list of data frames (`"data.frame"` class or derived) to be mapped to the `label` aesthetic. These geoms work with tibbles or data frames as `data` as they both support `list` objects as member variables.

A table is built with function `gridExtra::gtable` for each data frame in the list, and formatted according to a `ttheme` (table theme) list object or `ttheme` constructor function passed as argument to parameter `table.theme`. If the value passed as argument to `table.theme` is `NULL` the table theme used is that set as default through R option `ggpmisc.ttheme.default` at the time the plot is rendered or the `ttheme_gtdefault` constructor function if not set.

If the argument passed to `table.theme` or set through R option `ggpmisc.ttheme.default` 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` stored as a list object is passed as argument (e.g., by calling the constructor, using constructor name followed by parenthesis), it is used as is, with mappings to aesthetics `colour`, `fill`, `alpha`, and `family` 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 the values set in the `ttheme`. As the table is built with function `gridExtra::gtable()`, for details, please, consult `tableGrob` and `ttheme_gtdefault`.

The character strings in the data frame can be parsed into R expressions so the inset tables can include maths. With `parse = TRUE` parsing is attempted on each table cell, but failure triggers fallback to rendering without parsing, on a cell by cell basis. Thus, a table can contain a mixture cells and/or headings that require parsing or not (see the documentation in gridExtra-package for details).

The `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 inset table with respect to its *horizontal* and *vertical* axes (rows and columns in the data frame), and `angle` is used to rotate the inset table as a whole. The default for the `colour` and `fill` aesthetics is to retrieve them from the table theme, while `family` and `size` are retrieved from the `geom` element of the ggplot theme.

Of these two geoms only `geom_table` supports the plotting of connecting segments or arrows when its position has been modified by a `position` function. This is because `geom_table_npc` uses a coordinate system that is unrelated to data units, scales or data in other plot layers. In the case of `geom_table_npc`, `npcx` and `npcy` pseudo-aesthetics determine the position of the inset table. By default, `hjust` and `vjust` are set to `"inward"` to avoid clipping at the edge of the plot canvas.

Value

A plot layer instance.

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"`). In addition, you can use special alignments for justification including `"position"`, `"inward"` and `"outward"`. Inward always aligns text towards the center of the plotting area, and

outward aligns it away from the center of the plotting area. If tagged with `_mean` or `_median` (e.g., `"outward_mean"`) the mean or median of the data in the panel along the corresponding axis is used as center. If the characters following the underscore represent a number (e.g., `"outward_10.5"`) the reference point will be this value in data units. Position justification is computed based on the direction of the displacement of the position of the label so that each individual text or label is justified outwards from its original position. The default justification is `"position"`.

If no position displacement is applied, or a position function defined in 'ggplot2' is used, these geometries behave similarly to the corresponding ones from package 'ggplot2' with a default justification of `0.5` and no segment drawn.

Position functions

Many layer functions from package 'ggpp' are designed to work seamlessly with position functions that keep, rather than discard, the original `x` and `y` positions in `data` when computing a new displaced position. See `position_nudge_keep`, `position_dodge_keep`, `position_jitter_keep`, `position_nudge_center`, `position_nudge_line`, `position_nudge_to`, `position_dodgenudge`, `position_jitternudge`, and `position_stacknudge` for examples and details of their use.

Plot boundaries and clipping

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.

Note

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`.

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

Formatting of tables `stat_fmt_table`, `ttheme_gtdefault`, `ttheme_set`, `tableGrob`.

Other geometries adding layers with insets: `geom_plot()`

Examples

```
library(dplyr)
library(tibble)
```

```
theme_set(theme_bw())

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

df <- data.frame(x = 5.45, y = 34, tb = I(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))

# 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_gtstripes) +
  theme_classic()

# transparency
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df,
             aes(x = x, y = y, label = tb),
             alpha = 0.5) +
  theme_bw()

ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df,
             aes(x = x, y = y, label = tb),
             alpha = 0.5, alpha.target = "table.canvas")

df2 <- tibble(x = 5.45,
```

```
               y = c(34, 29, 24),
               x1 = c(2.29, 3.12, 4.00),
               y1 = c(26.6, 19.7, 15.1),
               cyl = c(4, 6, 8),
               tb = list(tb[1, 1:3], tb[2, 1:3], tb[3, 1:3]))

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

ggplot(mtcars,
       aes(wt, mpg, color = factor(cyl))) +
  geom_point() +
  geom_table(data = df2,
             inherit.aes = TRUE,
             colour.target = "table.rules",
             mapping = aes(x = x, y = y, label = tb))

# nudging and segments
ggplot(mtcars,
       aes(wt, mpg, color = factor(cyl))) +
  geom_point(show.legend = FALSE) +
  geom_table(data = df2,
             inherit.aes = TRUE,
             mapping = aes(x = x1, y = y1, label = tb),
             nudge_x = 0.7, nudge_y = 3,
             vjust = 0.5, hjust = 0.5,
             arrow = arrow(length = unit(0.5, "lines"))) +
  theme_classic()

ggplot(mtcars,
       aes(wt, mpg, color = factor(cyl))) +
  geom_point(show.legend = FALSE) +
  geom_table(data = df2,
             inherit.aes = TRUE,
             mapping = aes(x = x1, y = y1, label = tb),
             nudge_x = 0.7, nudge_y = 3,
             vjust = 0.5, hjust = 0.5,
             arrow = arrow(length = unit(0.5, "lines")),
             colour.target = c("table.rules", "segment")) +
  theme_classic()

# 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 highlight 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`. 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 combination 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.

Value

A plot layer instance.

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

Margin grobs can supplement a 2d display with annotations. Margin grobs such as icons or symbols can highlight individual 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`. 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.

Value

A plot layer instance.

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"). In addition, you can use special alignments for justification including "position", "inward" and "outward". Inward always aligns text towards the center of the plotting area, and outward aligns it away from the center of the plotting area. If tagged with _mean or _median (e.g., "outward_mean") the mean or median of the data in the panel along the corresponding axis is used as center. If the characters following the underscore represent a number (e.g., "outward_10.5") the reference point will be this value in data units. Position justification is computed based on the direction of the displacement of the position of the label so that each individual text or label is justified outwards from its original position. The default justification is "position".

If no position displacement is applied, or a position function defined in 'ggplot2' is used, these geometries behave similarly to the corresponding ones from package 'ggplot2' with a default justification of 0.5 and no segment drawn.

Position functions

Many layer functions from package 'ggpp' are designed to work seamlessly with position functions that keep, rather than discard, the original x and y positions in data when computing a new displaced position. See position_nudge_keep, position_dodge_keep, position_jitter_keep, position_nudge_center, position_nudge_line, position_nudge_to, position_dodgenudge, position_jitternudge, and position_stacknudge for examples and details of their use.

See Also

grid-package, geom_rug, and other documentation of package 'ggplot2'.

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

Margin points can supplement a 2d display with annotations. Margin points can highlight individual 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. 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 combination 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.

Value

A plot layer instance.

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.

Value

A "ggplot" object.

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

```
ggplot(lynx) + geom_line()
```

---

ivy.df *Ivy photosynthesis light response*

Description

A dataset containing photosynthesis measurements on four ivy plants.

Usage

```
ivy.df
```

Format

A `data.frame` object with 36 rows and 6 variables.

Details

For each plant a light response curve of photosynthesis was measured using a custom-built system and software that allowed controlling the concentrations of water vapour and carbon dioxide at the surface of the leaves, i.e., inside the air boundary layer.

References

Aphalo, P. J. (1991) Interactions in Stomatal Function. PhD thesis, University of Edinburgh. http://hdl.handle.net/1842/14758.

See Also

Other Plant growth and morphology data: `birch.df`

Examples

```
colnames(ivy.df)
head(ivy.df)
```

`position_dodgenudge` *Combined positions dodge and nudge*

Description

`position_dodgenudge()` combines into one function the action of `position_dodge` and `position_nudge` and `position_dodge2nudge()` combines into one function the action of `position_dodge2` and `position_nudge`. They are useful when labelling plots such as grouped bars, columns, etc. and when adding dodged text labels linked to observations plotted without dodge. It can replace other position functions as they are backwards compatible. Like all other position functions in 'ggpp' and 'ggrepel' they preserve the initial position to allow drawing of segments or arrow linking the original position to the displaced one.

Usage

```
position_dodgenudge(
  width = 1,
  preserve = c("total", "single"),
  reverse = FALSE,
  x = 0,
  y = 0,
  direction = c("none", "split", "split.x", "split.y", "center"),
  kept.origin = c("dodged", "original", "none")
)

position_dodge_keep(
  width = 1,
  preserve = c("total", "single"),
  reverse = FALSE,
  kept.origin = "original"
)

position_dodge2nudge(
  width = 1,
  preserve = c("total", "single"),
  padding = 0.1,
  reverse = FALSE,
  x = 0,
  y = 0,
  direction = c("none", "split", "split.x", "split.y", "center"),
  kept.origin = c("dodged", "original", "none")
)

position_dodge2_keep(
  width = 1,
  preserve = c("total", "single"),
  kept.origin = "original"
)
```

Arguments

width	Dodging width, when different to the width of the individual elements. This is useful when you want to align narrow geoms with wider geoms. See the examples.
preserve	Should dodging preserve the total width of all elements at a position, or the width of a single element?.
reverse	If TRUE, will reverse the default stacking order. This is useful if you're rotating both the plot and legend.
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', with nudge values in data rows order.

direction	One of "none", "split", "split.x" or "split.y". A value of "none" replicates the behavior of position_nudge. At the moment "split" changes the sign of the nudge at zero, which is suitable for column plots with negative slices.
kept.origin	One of "original", "dodged" or "none".
padding	Padding between elements at the same position. Elements are shrunk by this proportion to allow space between them. Defaults to 0.1.

Details

The applied dodge is identical to that by position_dodge or position_dodge2 while nudging is similar to that by position_nudge.

There are two possible uses for these functions. First they can be used to label dodged bars or boxplots. In this case, it is mandatory to use the same argument to width when passing position_dodge() to geom_col() and position_dodgenudge() to geom_text() or geom_label() or their repulsive equivalents. Otherwise the arrows or segments will fail to connect to the labels. In other words dodging is computed twice. Dodge is identical to that obtained with the same arguments in position_dodge as position_dodgenudge() last function simply calls the same code from package 'ggplot2' ahead of applying nudging.

The second use is to dodge labels to be connected to elements that have not been dodged.

When applying dodging, the return of original positions instead of the dodged ones is achieved by passing origin = "original" instead of the default of origin = "dodged".

Value

A "Position" object.

Author(s)

Michał Krassowski, edited by Pedro J. Aphalo.

Source

https://github.com/slowkow/ggrepel/issues/161.

See Also

position_nudge, position_nudge_repel.

Other position adjustments: position_dodgenudge_to(), position_jitternudge(), position_nudge_center(), position_nudge_keep(), position_nudge_line(), position_nudge_to(), position_stacknudge(), position_stacknudge_to()

Examples

```
df <- data.frame(x1 = c(1, 2, 1, 3, -1),
                 x2 = c("a", "a", "b", "b", "b"),
                 grp = c("some long name", "other name", "some name",
                         "another name", "some long name"))
```

```
# Add labels to a horizontal column plot (stacked by default)
ggplot(data = df, aes(x1, x2, group = grp)) +
  geom_col(aes(fill = grp), width = 0.8,
           position = position_dodge()) +
  geom_vline(xintercept = 0) +
  geom_text(
    aes(label = grp),
    position = position_dodgenudge(x = 0.09, direction = "split", width = 0.8),
    angle = 90, size = 3) +
  theme(legend.position = "none")

ggplot(data = df, aes(x2, x1, group = grp)) +
  geom_col(aes(fill = grp), width = 0.75,
           position = position_dodge(width = 0.75)) +
  geom_vline(xintercept = 0) +
  geom_text(aes(label = grp),
            position = position_dodgenudge(y = 0.1,
                                           direction = "split",
                                           width = 0.75),
            size = 3) +
  theme(legend.position = "none")
```

---

position_dodgenudge_to *Dodge plus nudge labels to new positions*

Description

Functions position_dodgenudge_to() and position_dodge2nudge_to() are meant to complement position_dodge() and position_dodge2() from 'ggplot2', adding as a second action that of position_nudge_to(). These positions are generally useful for adjusting the position of labels or text. As with other position functions in this package, the original positions are preserved to allow the text or labels to be linked back to their original position with a segment or arrow.

Usage

```
position_dodgenudge_to(
  width = 1,
  preserve = c("total", "single"),
  reverse = FALSE,
  x = NULL,
  y = NULL,
  x.action = c("none", "spread"),
  y.action = c("none", "spread"),
  x.distance = "equal",
  y.distance = "equal",
  x.expansion = 0,
  y.expansion = 0,
```

```
  kept.origin = c("dodged", "original", "none")
)

position_dodge2nudge_to(
  width = 1,
  preserve = c("total", "single"),
  padding = 0.1,
  reverse = FALSE,
  x = NULL,
  y = NULL,
  x.action = c("none", "spread"),
  y.action = c("none", "spread"),
  x.distance = "equal",
  y.distance = "equal",
  x.expansion = 0,
  y.expansion = 0,
  kept.origin = c("dodged", "original", "none")
)
```

Arguments

`width`	Dodging width, when different to the width of the individual elements. This is useful when you want to align narrow geoms with wider geoms. See the examples.
`preserve`	Should dodging preserve the total width of all elements at a position, or the width of a single element?.
`reverse`	If TRUE, will reverse the default dodging order.
`x, y`	Coordinates of the destination position. A vector of mode `numeric`, that is extended if needed, to the same length as rows there are in `data`. The default, `NULL`, leaves the original coordinates unchanged after dodging.
`x.action, y.action`	character string, one of `"none"`, or `"spread"`. With `"spread"` distributing the positions within the range of argument `x` or `y`, if non-null, or the range the variable mapped to *x* or `y`, otherwise.
`x.distance, y.distance`	character or numeric Currently only `"equal"` is implemented.
`x.expansion, y.expansion`	numeric vectors of length 1 or 2, as a fraction of width of the range.
`kept.origin`	One of `"original"`, `"dodged"` or `"none"`.
`padding`	Padding between elements at the same position. Elements are shrunk by this proportion to allow space between them. Defaults to 0.1.

Details

These positions apply sequentially two actions, in the order they appear in their names. The applied dodge is similar to that by `position_dodge` and `position_dodge2` while nudging is different to that by `position_nudge` and equal to that applied by `position_nudge_to`.

The dodged and nudged to x and/or y values replace the original ones in data, while the original or the dodged coordinates are returned in x_orig and y_orig. Nudge values supported are those of *mode* numeric, thus including dates and times when they match the mapped data.

If the length of x and/or y is more than one but less than rows are present in the data, the vector is both recycled and reordered so that the nudges are applied sequentially based on the data values. If their length matches the number of rows in data, they are assumed to be already in data order.

The intended use is to label dodged bars, boxplots or points with labels aligned. In this case, it is mandatory to use the same argument to width when passing position_dodge() to geom_col() and position_dodgenudge_to() to geom_text(), geom_label(), geom_text_s(), geom_label_s() or their repulsive equivalents from package 'ggrepel'. Otherwise the arrows or segments will fail to connect to the labels.

When applying dodging, the return of original positions instead of the dodged ones is achieved by passing origin = "original" instead of the default of origin = "dodged".

Value

A "Position" object.

Note

Irrespective of the action, the ordering of rows in data is preserved.

See Also

position_nudge_to, position_dodge, position_dodge2.

Other position adjustments: position_dodgenudge(), position_jitternudge(), position_nudge_center(), position_nudge_keep(), position_nudge_line(), position_nudge_to(), position_stacknudge(), position_stacknudge_to()

Examples

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

# default is no nudging
ggplot(df, aes(grp, y, label = label, fill = label)) +
  geom_col(position = position_dodge(width = 0.92)) +
  geom_text(position = position_dodgenudge_to(width = 0.92),
            vjust = -0.2) +
  theme(legend.position = "none")

ggplot(df, aes(grp, y, label = label, fill = grp.inner)) +
  geom_col(position = position_dodge(width = 0.92)) +
  geom_text(position = position_dodgenudge_to(width = 0.92),
            vjust = -0.2)
```

```
ggplot(df, aes(grp, y, label = label, fill = label)) +
  geom_col(position = position_dodge2(width = 0.92)) +
  geom_text(position = position_dodge2nudge_to(width = 0.92),
            vjust = -0.2) +
  theme(legend.position = "none")

ggplot(df, aes(grp, y, label = label, fill = grp.inner)) +
  geom_col(position = position_dodge2(width = 0.92)) +
  geom_text(position = position_dodge2nudge_to(width = 0.92),
            vjust = -0.2)

# nudging all labels to a given y value
ggplot(df, aes(grp, y, label = label, fill = grp.inner)) +
  geom_col(position = position_dodge(width = 0.92)) +
  geom_text(position = position_dodgenudge_to(width = 0.92, y = 0.8))

ggplot(df, aes(grp, y, label = label, fill = grp.inner)) +
  geom_col(position = position_dodge2(width = 0.92)) +
  geom_text(position = position_dodge2nudge_to(width = 0.92, y = 0.8))

ggplot(df, aes(grp, y, label = label, fill = grp.inner)) +
  geom_col(position = position_dodge(width = 0.92)) +
  geom_text(position = position_dodgenudge_to(width = 0.92, y = 0.8))

ggplot(df[-1, ], aes(grp, y, label = label, fill = grp.inner)) +
  geom_col(position = position_dodge(width = 0.92)) +
  geom_text(position = position_dodgenudge_to(width = 0.92, y = 0.8))

ggplot(df[-1, ], aes(grp, y, label = label, fill = grp.inner)) +
  geom_col(position = position_dodge(width = 0.92, preserve = "total")) +
  geom_text(position = position_dodgenudge_to(width = 0.92, y = 0.8,
                                              preserve = "total"))

ggplot(df[-1, ], aes(grp, y, label = label, fill = grp.inner)) +
  geom_col(position = position_dodge(width = 0.92, preserve = "single")) +
  geom_text(position = position_dodgenudge_to(width = 0.92, y = 0.8,
                                              preserve = "single"))

ggplot(df[-1, ], aes(grp, y, label = label, fill = grp.inner)) +
  geom_col(position = position_dodge2(width = 0.92, preserve = "total")) +
  geom_text(position = position_dodge2nudge_to(width = 0.92, y = 0.8,
                                               preserve = "total"))

ggplot(df[-1, ], aes(grp, y, label = label, fill = grp.inner)) +
  geom_col(position = position_dodge2(width = 0.92, preserve = "single")) +
  geom_text(position = position_dodge2nudge_to(width = 0.92, y = 0.8,
                                               preserve = "single"))
```

`position_jitternudge` *Combined positions jitter and nudge*

Description

position_jitternudge() combines into one function the action of position_jitter and position_nudge. It is useful when labels to jittered plots and when adding jitter to text labels linked to points plotted without jitter. It can replace other position functions as it is backwards compatible. Like all other position functions in 'ggpp' and 'ggrepel' it preserves the initial position to allow drawing of segments or arrow linking the original position to the displaced one.

Usage

```
position_jitternudge(
  width = NULL,
  height = NULL,
  seed = NA,
  x = 0,
  y = 0,
  direction = c("as.is", "alternate", "split"),
 nudge.from = c("original", "original.x", "original.y", "jittered", "jittered.y",
    "jittered.x"),
  kept.origin = c("jittered", "original", "none")
)

position_jitter_keep(
  width = NULL,
  height = NULL,
  seed = NA,
  kept.origin = "original"
)
```

Arguments

width, height	Amount of vertical and horizontal jitter. The jitter is added in both positive and negative directions, so the total spread is twice the value specified here. If omitted, defaults to 40 resolution of the data: this means the jitter values will occupy 80 implied bins. Categorical data is aligned on the integers, so a width or height of 0.5 will spread the data so it's not possible to see the distinction between the categories.
seed	A random seed to make the jitter reproducible. Useful if you need to apply the same jitter twice, e.g., for a point and a corresponding label. The random seed is reset after jittering. If NA (the default value), the seed is initialised with a random value; this makes sure that two subsequent calls start with a different seed. Use NULL to use the current random seed and also avoid resetting (the behaviour of ggplot 2.2.1 and earlier).
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, with nudge values in data rows order.
direction	One of "as.is", "alternate", "split", "split.x" or "split.y". A value of "none" replicates the behavior of position_nudge. "split" changes the

sign of the nudge depending on the direction of the random jitter applied to each individual observation, which is suitable for nudging labels outward of the jittered data.

`nudge.from` One of `"original"`, `"jittered"`, `"original.y"` (or `"jittered.x"`), `"original.x"` (or `"jittered.y"`). A value of `"original"` applies the nudge before jittering the observations, while `"jittered"` applies the nudging after jittering.

`kept.origin` One of `"original"`, `"jittered"` or `"none"`.

Details

Jitter with `position_jitternudge()` is identical to that with `position_jitter` while nudging is enhanced compared to `position_nudge` by taking into use cases specific to the combination of jitter and nudge.

There are two posible uses for this function. First it can be used to label jittered points in a plot. In this case, it is mandatory to use the same arguments to `width`, `height` and `seed` when passing `position_jitter()` to `geom_point()` and `position_jitternudge()` to `geom_text()` or to `geom_label()` or their repulsive equivalents. Otherwise the arrows or segments will fail to connect to the labels. In other words jittering is computed twice. Jitter should be identical with the same arguments as `position_jitternudge()` as this last function calls the same code imported from package 'ggplot2'.

The second use is to jitter labels to be connected to points that have not been jittered. The return of original positions instead of the jittered ones is achieved by passing `origin = "original"` to override the default `origin = "jittered"`.

Value

A `"Position"` object. The layer function within it returns a data frame, with the jittered + nudged values in columns `x` and `y` and by default the jittered values with no nudging as `x_orig` and `y_orig`. With `nudge.from = "original"` the original values with no jitter and no nudge applied are returned as `x_orig` and `y_orig`.

Note

When `direction = "split"` is used together with no jitter, the split to left and right, or up and down is done at random.

Author(s)

Michał Krassowski, edited by Pedro J. Aphalo.

Source

https://github.com/slowkow/ggrepel/issues/161.

See Also

`position_jitter`, `position_nudge`, `position_nudge_repel`.

Other position adjustments: position_dodgenudge(), position_dodgenudge_to(), position_nudge_center(), position_nudge_keep(), position_nudge_line(), position_nudge_to(), position_stacknudge(), position_stacknudge_to()

Examples

```
jitter <- position_jitter(width = 0.2, height = 2, seed = 123)

jitter_nudge <- position_jitternudge(width = 0.2, height = 2,
                                     seed = 123, x = 0.1,
                                     direction = "split",
                                     nudge.from = "jittered")
ggplot(mpg[1:20, ],
       aes(cyl, hwy, label = drv)) +
  geom_point(position = jitter) +
  geom_text_s(position = jitter_nudge)

jitter_nudge <- position_jitternudge(width = 0.2, height = 2,
                                     seed = 123, x = 0.35,
                                     direction = "split",
                                     nudge.from = "original.x")
ggplot(mpg[1:20, ],
       aes(cyl, hwy, label = drv)) +
  geom_point(position = jitter) +
  geom_text_s(position = jitter_nudge)

jitter <- position_jitter(width = 0, height = 2, seed = 123)

jitter_nudge <- position_jitternudge(width = 0, height = 2,
                                     seed = 123, x = 0.4,
                                     direction = "split",
                                     nudge.from = "original.x")
ggplot(mpg[1:20, ],
       aes(cyl, hwy, label = drv)) +
  geom_point(position = jitter) +
  geom_text_s(position = jitter_nudge)

jitter_nudge <- position_jitternudge(width = 0, height = 2,
                                     seed = 123, x = 0.4,
                                     direction = "alternate",
                                     nudge.from = "original.x")
ggplot(mpg[1:20, ],
       aes(cyl, hwy, label = drv)) +
  geom_point(position = jitter) +
  geom_text_s(position = jitter_nudge)

# No nudge, show how points have moved with jitter

ggplot(mpg[1:20, ],
       aes(cyl, hwy, label = drv)) +
  geom_point() +
  geom_point_s(position =
               position_jitter_keep(width = 0.3, height = 2, seed = 123),
```

```
                color = "red",
                arrow = grid::arrow(length = unit(0.4, "lines")))
```

---

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 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,
  kept.origin = c("original", "none")
)

position_nudge_centre(
  x = 0,
  y = 0,
  center_x = NULL,
  center_y = NULL,
  direction = NULL,
  obey_grouping = NULL,
  kept.origin = c("original", "none")
)
```

Arguments

x, y	Amount of vertical and horizontal distance to move. A numeric vector, that is recycled if shorter than the number of rows 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 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.
kept.origin	One of "original" or "none".

Details

This position function is backwards compatible with 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*.

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 to *x* and *y* coordinates.

Value

A "Position" object.

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. Whendirection = "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 adjusted to ensure these segments are of the same lengths as those at other angles.

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

See Also

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

Other position adjustments: position_dodgenudge(), position_dodgenudge_to(), position_jitternudge(), position_nudge_keep(), position_nudge_line(), position_nudge_to(), position_stacknudge(), position_stacknudge_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_s(hjust = "left", vjust = "bottom",
              position = position_nudge(x = 0.2, y = 0.2))

ggplot(df, aes(x, y, label = y)) +
  geom_point() +
  geom_text_s(add.segments = FALSE,
              position = position_nudge_center(x = 0.2, y = 0.2)
  )

# "split" nudging

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text_s(aes(label = y),
              add.segments = FALSE,
              position = position_nudge_center(x = 0.2,
                                               y = 0.2,
                                               direction = "split"))

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text_s(aes(label = y),
              position = position_nudge_center(x = 0.4,
                                               direction = "split"))

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text_s(aes(label = y),
              position = position_nudge_center(y = 0.2,
                                               direction = "split"))

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text_s(aes(label = y),
              position = position_nudge_center(x = 0.2,
                                               y = 0.3,
                                               center_y = 2,
                                               center_x = 1.5,
                                               direction = "split"))

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text_s(aes(label = y),
              position = position_nudge_center(x = 0.06,
                                               y = 0.08,
                                               center_y = 2))
```

```
ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text_s(aes(label = y),
              position = position_nudge_center(x = 0.1,
                                               center_x = 2.51))

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text_s(aes(label = y),
              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_s(aes(label = y),
              position = position_nudge_center(x = 0.1,
                                               y = 0.2,
                                               direction = "radial"))

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text_s(aes(label = y),
              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_s(aes(label = y),
              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_s(aes(label = y),
              position = position_nudge_center(x = -1.2,
                                               y = -3,
```

```
                                          center_x = 0,
                                          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,
                                             obey_grouping = FALSE))
```

`position_nudge_keep` *Nudge points a fixed distance*

Description

The function `position_nudge_keep()` has an additional parameters compared to `position_nudge`, `obey_grouping` and by default the same behaviour when the values passed as arguments to `x` and `y` have length one.

Usage

```
position_nudge_keep(
  x = 0,
  y = 0,
  obey_grouping = NULL,
  kept.origin = c("original", "none")
)
```

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`, with nudge values in data rows order.
`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.
`kept.origin`	One of `"original"` or `"none"`.

Details

When `x` or `y` have length > 1, they are treated specially. If the lengths is the same as there are rows in data, the nudges are applied in the order of the rows in data. When they are shorter, they are recycled and applied to the data values after ordering. This makes it possible to have alternating mudging

right and left or up and down. If `obey_grouping = TRUE` is passed in the call, the alternation will take place within groups.

As other position functions from package 'ggpp', `position_nudge_keep()` by default renames and keeps the original positions of the observations in `data` making it possible to draw connecting segments or conencting arrows.

Value

A `"Position"` object.

Note

Irrespective of the action, the ordering of rows in `data` is preserved.

See Also

Other position adjustments: `position_dodgenudge()`, `position_dodgenudge_to()`, `position_jitternudge()`, `position_nudge_center()`, `position_nudge_line()`, `position_nudge_to()`, `position_stacknudge()`, `position_stacknudge_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_s(hjust = "left", vjust = "bottom",
              position = position_nudge_keep(x = 0.2, y = 0.2))

# alternating nudging
ggplot(df, aes(x, y, label = y)) +
  geom_point() +
  geom_text_s(position = position_nudge_keep(x = c(0.2, -0.2)))

# direct nudging
ggplot(df, aes(x, y, label = y)) +
  geom_point() +
  geom_text_s(position = position_nudge_keep(x = rep_len(c(0.2, -0.2), 6)))
```

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 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 = c("automatic", "none", "split"),
  line_nudge = 1,
  kept.origin = c("original", "none")
)
```

Arguments

x, y	Amount of vertical and horizontal distance to move. A numeric vector of length 1 or longer.
xy_relative	Nudge relative to *x* and *y* data expanse, ignored unless x and y are both NAs.
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 "automatic", "none", or "split".
line_nudge	A positive multiplier >= 1, increasing nudging away from the curve or line compared to nudging from points.
kept.origin	One of "original" or "none".

Details

The default amount 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 or 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"`.

Value

A `"Position"` object.

Note

For `method = "lm"` only model formulas corresponding to polynomials with no missing terms are supported. If using`poly` in the model formula, `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

`position_nudge`, `position_nudge_repel`.

Other position adjustments: `position_dodgenudge()`, `position_dodgenudge_to()`, `position_jitternudge()`, `position_nudge_center()`, `position_nudge_keep()`, `position_nudge_to()`, `position_stacknudge()`, `position_stacknudge_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_s(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, 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(subset(df, x >= 0), aes(y, sqrt(y), label = l)) +
  geom_line(linetype = "dotted") +
  geom_point() +
  geom_text(position = position_nudge_line())

# Points scattered near a curve or line, we use '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)))
```

`position_nudge_to` *Nudge labels to new positions*

Description

`position_nudge_to()` differs from `position_nudge` in that the coordinates of the new position are given directly, rather than as a displacement from the original location. It optionally sets an even spacing among positions within a range. As with other position functions in this package, the original positions are preserved to allow the text or labels to be linked back to their original position with a segment or arrow.

Usage

```
position_nudge_to(
  x = NULL,
  y = NULL,
  x.action = c("none", "spread"),
  y.action = c("none", "spread"),
  x.distance = "equal",
  y.distance = "equal",
  x.expansion = 0,
  y.expansion = 0,
  kept.origin = c("original", "none")
)
```

Arguments

`x, y`	Coordinates of the destination position. A vector of mode `numeric`, that is extended if needed, to the same length as rows there are in `data`. The default, `NULL`, leaves the original coordinates unchanged.

`x.action`, `y.action`
: character string, one of `"none"`, or `"spread"`. With `"spread"` distributing the positions within the range of argument `x` or `y`, if non-null. Otherwise, using the range the variable mapped to *x* or `y`.

`x.distance`, `y.distance`
: character or numeric Currently only `"equal"` is implemented, indicating equal spacing between the spread positions.

`x.expansion`, `y.expansion`
: numeric vectors of length 1 or 2, as a fraction of width of the range used to spread positions.

`kept.origin`
: One of `"original"` or `"none"`.

Details

The nudged to `x` and/or `y` values replace the original ones in `data`, while the original coordinates are returned in `x_orig` and `y_orig`. Nudge values supported are those of *mode* numeric, thus including dates and times when they match the mapped data.

If the length of `x` and/or `y` is more than one but less than the rows present in the `data`, the vector is both recycled and reordered so that the nudges are applied sequentially based on the data values. If their length matches the number of rows in `data`, they are assumed to be already in `data` order.

Irrespective of the action, the ordering of rows in `data` is preserved.

Value

A `"Position"` object.

Note

The current implementation DOES NOT support flipping geoms with the `orientation` argument or implicitly by the mapping. It DOES NOT apply scale transformations when spreading the positions.

See Also

`position_nudge`, `position_nudge_repel`.

Other position adjustments: `position_dodgenudge()`, `position_dodgenudge_to()`, `position_jitternudge()`, `position_nudge_center()`, `position_nudge_keep()`, `position_nudge_line()`, `position_stacknudge()`, `position_stacknudge_to()`

Examples

```
# The examples below exemplify the features of position_nudge_to().
# Please see the vignette for examples of use cases.

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

```
)

# default is no nudging
ggplot(df, aes(label, y, label = y)) +
  geom_col() +
  geom_text(position = position_nudge_to(),
            vjust = -0.2)

# a single y (or x) value nudges all observations to this data value
ggplot(df, aes(label, y, label = y)) +
  geom_col() +
  geom_label(position = position_nudge_to(y = 1))

ggplot(df, aes(x, y, label = label)) +
  geom_point() +
  geom_text(position = position_nudge_to(y = 3.2))

ggplot(df, aes(x, y, label = label)) +
  geom_point() +
  geom_line() +
  geom_text(position = position_nudge_to(y = 0.1))

# with a suitable geom, segments or arrows can be added
ggplot(df, aes(x, y, label = label)) +
  geom_point() +
  geom_text_s(position = position_nudge_to(y = 2.25))

# alternating in y value order because y has fewer values than rows in data
ggplot(df, aes(x, y, label = label)) +
  geom_point() +
  geom_text_s(position = position_nudge_to(y = c(3, 0.1)))

# in data row order with as many nudge y values as rows in data
ggplot(df, aes(x, y, label = label)) +
  geom_point() +
  geom_text_s(position = position_nudge_to(y = c(1.8, 2.3, 1.3, 2.8, 3, 0.1)))

# spread the values at equal distance within the available space
ggplot(df, aes(x, y, label = label)) +
  geom_point() +
  geom_text_s(position =
                position_nudge_to(y = 4, x.action = "spread"))

ggplot(df, aes(x, y, label = label)) +
  geom_point() +
  geom_text_s(position =
                position_nudge_to(y = 4, x.action = "spread")) +
scale_x_log10()

# spread the values at equal distance within the expanded available space
ggplot(df, aes(x, y, label = label)) +
  geom_point() +
  geom_text_s(position =
```

```
                position_nudge_to(y = 4, x.action = "spread", x.expansion = 0.1))

# spread the values at equal distance within the range given by x
ggplot(df, aes(x, y, label = label)) +
  geom_point() +
  geom_text_s(position =
                position_nudge_to(y = 4, x = c(1.5, 4), x.action = "spread"))

# currently if scale transformations are used, the x and/or y arguments must
# be transformed. WARNING: This will change in the near future!!

ggplot(df, aes(x, y, label = label)) +
  geom_point() +
  geom_text_s(position =
                position_nudge_to(y = 4, x = log10(c(1.5, 4)), x.action = "spread")) +
scale_x_log10()

ggplot(df, aes(x, y, label = label)) +
  geom_point() +
  geom_text_s(position =
                position_nudge_to(y = log10(4), x.action = "spread")) +
scale_y_log10()
```

---

position_stacknudge *Combined positions stack and nudge*

Description

position_stacknudge() is useful when labelling plots such as stacked bars, stacked columns, stacked lines, etc. In contrast to position_nudge, position_stacknudge() returns in data both the original coordinates and the nudged coordinates.

Usage

```
position_stacknudge(
  vjust = 1,
  reverse = FALSE,
  x = 0,
  y = 0,
  direction = c("none", "split", "split.x", "split.y"),
  kept.origin = c("stacked", "original", "none")
)

position_fillnudge(
  vjust = 1,
  reverse = FALSE,
  x = 0,
  y = 0,
```

```
  direction = c("none", "split", "split.x", "split.y"),
  kept.origin = c("stacked", "original", "none")
)

position_stack_keep(vjust = 1, reverse = FALSE, kept.origin = "original")

position_fill_keep(vjust = 1, reverse = FALSE, kept.origin = "original")

position_stack_minmax(
  vjust = 1,
  reverse = FALSE,
  x = 0,
  y = 0,
  direction = c("none", "split", "split.x", "split.y"),
  kept.origin = c("stacked", "original", "none")
)
```

Arguments

vjust	Vertical adjustment for geoms that have a position (like points or lines), not a dimension (like bars or areas). Set to 0 to align with the bottom, 0.5 for the middle, and 1 (the default) for the top.
reverse	If TRUE, will reverse the default stacking order. This is useful if you're rotating both the plot and legend.
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, with nudge values in data rows order.
direction	One of "none", "split", "split.x" or "split.y". A value of "none" replicates the behavior of position_nudge. At the moment "split" changes the sign of the nudge at zero, which is suitable for column plots with negative slices.
kept.origin	One of "original", "stacked" or "none".

Details

position_fillnudge() is useful when labelling plots such as filled bars, filled columns, filled lines, etc. In contrast to position_nudge, position_fillnudge() returns in data both the original coordinates and the nudged coordinates.

The wrapper position_nudge_keep() has the same signature and behaviour as position_nudge nad provides an easier to remember name when the need is only to have access to both the original and nudged coordinates.

These position functions are backwards compatible with position_nudge but extends it by adding support for stacking and for geometries that make use of the original position to draw connecting segments or arrows.

The wrapper position_stack_keep() has the same signature and behaviour as position_stack and provides an easier to remember name when the need is only to have access to both the original and nudged coordinates.

The wrapper position_fill_keep() has the same signature and behaviour as position_fill and provides an easier to remember name when the need is only to have access to both the original and nudged coordinates.

The wrapper position_stack_minmax() has the same signature and behaviour as position_stacknudge but stacks y, ymin and ymax in parallel, making it possible to stack summaries with error bars, works correctly with geom_pointrange(), geom_linerange() and geom_errorbar().

Value

A "Position" object.

Author(s)

Michał Krassowski, edited by Pedro J. Aphalo.

Source

https://github.com/slowkow/ggrepel/issues/161.

See Also

position_nudge, position_stack, position_nudge_repel.

Other position adjustments: position_dodgenudge(), position_dodgenudge_to(), position_jitternudge(), position_nudge_center(), position_nudge_keep(), position_nudge_line(), position_nudge_to(), position_stacknudge_to()

Examples

```
df <- data.frame(x1 = c("a", "a", "b", "b", "b"),
                 x2 = c(1, 2, 1, 3, -1),
                 grp = c("some long name", "other name", "some name",
                         "another name", "some long name"))

# Add labels to a horizontal column plot (stacked by default)
ggplot(data = df, aes(x1, x2, group = grp)) +
  geom_col(aes(fill = grp), width=0.5) +
  geom_vline(xintercept = 0) +
  geom_text(
    aes(label = grp),
    position = position_stacknudge(vjust = 0.5, y = 0.3)) +
  theme(legend.position = "none")

# Add labels to a vertical column plot (stacked by default)
ggplot(data = df, aes(x2, x1, group = grp)) +
  geom_col(aes(fill = grp), width=0.5) +
  geom_vline(xintercept = 0) +
  geom_text(
    aes(label = grp),
    position = position_stacknudge(vjust = 0.5, x = -0.3),
    angle = 90) +
  theme(legend.position = "none")
```

```
# Add labels to a vertical column plot (stacked by default)
ggplot(data = subset(df, x1 >= 0), aes(x1, x2, group = grp)) +
  geom_col(aes(fill = grp), width=0.5, position = position_fill()) +
  geom_vline(xintercept = 0) +
  geom_text(
    aes(label = grp),
    position = position_fillnudge(vjust = 0.5, x = -0.3),
    angle = 90) +
  theme(legend.position = "none")

# Add label at a fixed distance from the top of each column slice
ggplot(data = df, aes(x1, x2, group = grp)) +
  geom_col(aes(fill = grp), width=0.5) +
  geom_vline(xintercept = 0) +
  geom_text(
    aes(label = grp),
    position = position_stacknudge(vjust = 1, y = -0.2)) +
  theme(legend.position = "none")

# Use geom_text_s(), geom_text_repel() or geom_label_repel() to link
# label to labelled segment or object with an arrow
ggplot(data = df, aes(x2, x1, group = grp)) +
  geom_col(aes(fill = grp), width=0.5) +
  geom_vline(xintercept = 0) +
  geom_text_s(
    aes(label = grp),
    position = position_stacknudge(vjust = 0.5, y = 0.35),
    vjust = "bottom") +
  theme(legend.position = "none")

ggplot(birch_dw.df,
       aes(y = dry.weight * 1e-3, x = Density, fill = Part)) +
   stat_summary(geom = "col", fun = mean,
                position = "stack", alpha = 0.7, width = 0.67) +
   stat_summary(geom = "linerange", fun.data = mean_cl_normal,
                position = position_stack_minmax()) +
  labs(y = "Seedling dry mass (g)") +
  scale_fill_grey(start = 0.7, end = 0.3) +
  facet_wrap(facets = vars(Container))
```

`position_stacknudge_to` *Stack plus nudge labels to new positions*

Description

Functions `position_stacknudge_to()` and `position_fillnudge_to()` are meant to complement `position_stack()` and `position_fill()` from 'ggplot2', adding as a second action that

of `position_nudge_to()`. These positions are generally useful for adjusting the position of labels or text. As with other position functions in this package, the original positions are preserved to allow the text or labels to be linked back to their original position with a segment or arrow.

Usage

```
position_stacknudge_to(
  vjust = 1,
  reverse = FALSE,
  x = NULL,
  y = NULL,
  x.action = c("none", "spread"),
  y.action = c("none", "spread"),
  x.distance = "equal",
  y.distance = "equal",
  x.expansion = 0,
  y.expansion = 0,
  kept.origin = c("stacked", "original", "none")
)

position_fillnudge_to(
  vjust = 1,
  reverse = FALSE,
  x = NULL,
  y = NULL,
  x.action = c("none", "spread"),
  y.action = c("none", "spread"),
  x.distance = "equal",
  y.distance = "equal",
  x.expansion = 0,
  y.expansion = 0,
  kept.origin = c("stacked", "original", "none")
)
```

Arguments

`vjust`	Vertical adjustment for geoms that have a position (like points or lines), not a dimension (like bars or areas). Set to 0 to align with the bottom, 0.5 for the middle, and 1 (the default) for the top.
`reverse`	If TRUE, will reverse the default stacking order. This is useful if you're rotating both the plot and legend.
`x, y`	Coordinates of the destination position. A vector of mode `numeric`, that is extended if needed, to the same length as rows there are in `data`. The default, `NULL`, leaves the original coordinates unchanged after dodging.
`x.action, y.action`	character string, one of `"none"`, or `"spread"`. With `"spread"` distributing the positions within the range of argument `x` or `y`, if non-null, or the range the variable mapped to *x* or `y`, otherwise.

`x.distance`, `y.distance`
: character or numeric Currently only `"equal"` is implemented.

`x.expansion`, `y.expansion`
: numeric vectors of length 1 or 2, as a fraction of width of the range.

`kept.origin`
: One of `"original"`, `"stacked"` or `"none"`.

Details

These positions apply sequentially two actions, in the order they appear in their names. The applied stacking is similar to that by `position_stack` and `position_fill` while nudging is different to that by `position_nudge` and equal to that applied by `position_nudge_to`.

The nudged to `x` and/or `y` values replace the original ones in `data`, while the original or the stacked coordinates are returned in `x_orig` and `y_orig`. Nudge values supported are those of *mode* numeric, thus including dates and times when they match the mapped data.

If the length of `x` and/or `y` is more than one but less than rows are present in the data, the vector is both recycled and reordered so that the nudges are applied sequentially based on the data values. If their length matches the number of rows in data, they are assumed to be already in data order.

When applying stacking, the return of original positions instead of the stacked ones is achieved by passing `origin = "original"` instead of the default of `origin = "stacked"`.

Value

A `"Position"` object.

Note

Irrespective of the action, the ordering of rows in `data` is preserved.

See Also

`position_nudge_to`, `position_stack`.

Other position adjustments: `position_dodgenudge()`, `position_dodgenudge_to()`, `position_jitternudge()`, `position_nudge_center()`, `position_nudge_keep()`, `position_nudge_line()`, `position_nudge_to()`, `position_stacknudge()`

Examples

```
cols.df <- data.frame(x1 = c(1, 2, 1, 3),
                      x2 = c("a", "a", "b", "b"),
                      grp = c("A", "B", "C", "D"))

 ggplot(data = cols.df, aes(x2, x1, group = grp)) +
 geom_col(aes(fill = grp), width=0.5, position = "stack") +
   geom_vline(xintercept = 0) +
   geom_text_s(
     aes(label = x1),
     position = position_stacknudge_to(x = c(1.4, 1.4, 1.6, 1.6))) +
   theme(legend.position = "none")
```

```
ggplot(data = cols.df, aes(x2, x1, group = grp)) +
geom_col(aes(fill = grp), width=0.5, position = "fill") +
  geom_vline(xintercept = 0) +
  geom_text_s(
    aes(label = grp),
    position = position_fillnudge_to(x = c(1.4, 1.4, 1.6, 1.6))) +
  theme(legend.position = "none")

ggplot(data = cols.df, aes(x2, x1, group = grp)) +
geom_col(aes(fill = grp), width=0.5, position = "fill") +
  geom_vline(xintercept = 0) +
  geom_text_s(
    aes(label = x1),
    position = position_fillnudge_to(vjust = 0.5,
                                     x = c(1.4, 1.4, 1.6, 1.6))) +
  theme(legend.position = "none")

ggplot(data = cols.df, aes(x2, x1, group = grp)) +
geom_col(aes(fill = grp), width=0.5, position = "fill") +
  geom_vline(xintercept = 0) +
  geom_text_s(
    aes(label = x1), vjust = 0.5,
    position = position_fillnudge_to(vjust = 0.5,
                                     x = c(1.4, 1.4, 1.6, 1.6),
                                     y = c(0.85, 0.35, 0.85, 0.35))) +
  theme(legend.position = "none")
```

`quadrant_example.df` *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

References

Rai, Neha; O'Hara, Andrew; Farkas, Daniel; Safronov, Omid; Ratanasopa, Khuanpiroon; Wang, Fang; Lindfors, Anders V.; Jenkins, Gareth I.; Lehto, Tarja; Salojärvi, Jarkko; Brosché, Mikael; Strid. Åke; Aphalo, Pedro José; Morales, Luis Orlando (2020) The photoreceptor UVR8 mediates the perception of both UV-B and UV-A wavelengths up to 350 nm of sunlight with responsivity moderated by cryptochromes. *Plant, Cell & Environment*, 43:1513-1527.

See Also

Other Transcriptomics data: volcano_example.df

Examples

```
colnames(quadrant_example.df)
head(quadrant_example.df)
```

---

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()

Value

A "Scale" object.

---

stat_apply_group *Apply a function to x or y values*

Description

stat_summary_xy() and stat_centroid() are similar to ggplot2::stat_summary() but summarize both x and y values in the same plot layer. Differently to stat_summary() no grouping based on data values is done; the grouping respected is that already present based on mappings to aesthetics. This makes it possible to highlight the actual location of the centroid with geom_point(), geom_text(), and similar geometries. Instead, if we use geom_rug() they are only a convenience avoiding the need to add two separate layers and flipping one of them using orientation = "y".

Usage

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

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

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

stat_apply_group applies functions to data. When possible it is preferable to use transformations through scales or summary functions such as ggplot2::stat_summary(), stat_summary_xy() or stat_centroid(). There are some computations that are not scale transformations but are not usual summaries either, as the number of data values does not decrease all the way to one row per group. A typical case for a summary is the computation of quantiles. For transformations are cumulative ones, e.g., using cumsum(), runmed() and similar functions. Obviously, it is always possible to apply such functions to the data before plotting and passing them to a single layer function. However, it can be useful 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 to these statistics, geom_smooth applies a function that takes both x and y values as arguments.

These four statistics are similar. They differ on whether they return a single or multiple rows of data per group.

Value

A data frame with the same variables as the data input, with either a single or multiple rows, with the values of x and y variables replaced by the values returned by the applied functions, or possibly filled with NA if no function was supplied or available by default. If the applied function returns a named vector, the names are copied into columns x.names and/or y.names. If the summary function applied returns a one row data frame, it will be column bound keeping the column names, but overwritting columns x and/or y with y from the summary data frame. In the names returned by .fun.x the letter "y" is replaced by "x". These allows the use of the same functions as in ggplot2::stat_summary().

x x-value as returned by .fun.x, with names removed

y y-value as returned by `.fun.y`, with names removed

x.names if the x-value returned by `.fun.x` is named, these names

y.names if the y-value returned by `.fun.y` is named, these names

xmin, xmax values returned by `.fun.x` under these names, if present

ymin, ymax values returned by `.fun.y` under these names, if present

<other> additional values as returned by `.fun.y` under other names

Note

The applied function(s) must accept as first argument a vector that matches the variables mapped to `x` or `y` aesthetics. For `stat_summary_xy()` and `stat_centroid()` the function(s) to be applied is(are) expected to return a vector of length 1 or a data frame with only one row, as `mean_se()`, `mean_cl_normal()` `mean_cl_boot()`, `mean_sdl()` and `median_hilow()` from 'ggplot2' do.

For `stat_apply_group` the vectors returned by the the functions applied to `x` and `y` must be of exactly the same length. When only one of `.fun.x` or `.fun.y` are passed a function as argument, the other variable in the returned data is filled with `NA_real_`. If other values are desired, they can be set by means of a user-defined function.

References

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

Examples

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

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

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

ggplot(my.df, aes(x = X, y = Y, colour = category)) +
  stat_centroid(geom = "rug", linewidth = 1.5, .fun = median) +
  geom_point()

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

ggplot(my.df, aes(x = X, y = Y, colour = category)) +
  stat_summary_xy(geom = "pointrange",
                  .fun.x = mean, .fun.y = mean_se) +
```

```
  geom_point()

# quantiles
ggplot(my.df, aes(x = X, y = Y, colour = category)) +
  geom_point() +
  stat_apply_group(geom = "rug", .fun.y = quantile, .fun.x = quantile)

ggplot(my.df, aes(x = X, y = Y)) +
  geom_point() +
  stat_apply_group(geom = "rug", sides = "lr", color = "darkred",
                   .fun.y = quantile) +
  stat_apply_group(geom = "text", hjust = "right", color = "darkred",
                   .fun.y = quantile,
                   .fun.x = function(x) {rep(22, 5)}, # set x to 22
                   mapping = aes(label = after_stat(y.names))) +
                   expand_limits(x = 21)

my.probs <- c(0.25, 0.5, 0.75)
ggplot(my.df, aes(x = X, y = Y, colour = category)) +
  geom_point() +
  stat_apply_group(geom = "hline",
                   aes(yintercept = after_stat(y)),
                   .fun.y = quantile,
                   .fun.y.args = list(probs = my.probs))

# cummulative summaries
ggplot(my.df, aes(x = X, y = Y, colour = category)) +
  stat_apply_group(.fun.x = function(x) {x},
                   .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.x = function(x) {x[-1L]},
                   .fun.y = diff, na.rm = TRUE)

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

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

```
  ggplot(my.df, aes(x = X, y = Y, colour = category)) +
    stat_centroid(.fun = mean_se, geom = "debug_group")

  ggplot(my.df, aes(x = X, y = Y, colour = category)) +
    stat_summary_xy(.fun.y = mean_se, geom = "debug_group")

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

  ggplot(my.df, aes(x = X, y = Y, colour = category)) +
    geom_point() +
    stat_apply_group(geom = "debug_group",
                     .fun.x = quantile,
                     .fun.x.args = list(probs = my.probs),
                     .fun.y = quantile,
                    .fun.y.args = list(probs = my.probs))
}
```

`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. By default the data are handled all together, but it is also possible to control labeling separately in each tail.

Usage

```
stat_dens1d_filter(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  keep.fraction = 0.1,
  keep.number = Inf,
  keep.sparse = TRUE,
  keep.these = FALSE,
  exclude.these = FALSE,
  these.target = "label",
  pool.along = c("x", "none"),
  xintercept = 0,
  invert.selection = FALSE,
```

```
  bw = "SJ",
  kernel = "gaussian",
  adjust = 1,
  n = 512,
  return.density = FALSE,
  orientation = c("x", "y"),
  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,
  keep.these = FALSE,
  exclude.these = FALSE,
  these.target = "label",
  pool.along = c("x", "none"),
  xintercept = 0,
  invert.selection = FALSE,
  na.rm = TRUE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  bw = "SJ",
  adjust = 1,
  kernel = "gaussian",
  n = 512,
  return.density = FALSE,
  orientation = c("x", "y"),
  ...
)
```

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 vector of length 1 or 2 [0..1]. The fraction of the observations (or rows) in data to be retained.

keep.number integer vector of length 1 or 2. 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.

keep.these, exclude.these
character vector, integer vector, logical vector or function that takes one or more variables in data selected by these.target. Negative integers behave as in R's extraction methods. The rows from data indicated by keep.these and exclude.these are kept or excluded irrespective of the local density.

these.target character, numeric or logical selecting one or more column(s) of data. If TRUE the whole data object is passed.

pool.along character, one of "none" or "x", indicating if selection should be done pooling the observations along the *x* aesthetic, or separately on either side of xintercept.

xintercept numeric The split point for the data filtering. If NA the data are not split.

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.

return.density logical vector of lenght 1. If TRUE add columns "density" and "keep.obs" to the returned data frame.

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.

Details

The 1D density of observations of *x* or *y* is computed with function density and used to select observations, passing to the geom a subset of the rows in its data input. The default is to select observations in sparse regions of the plot, but the selection can be inverted so that only observations in the densest regions are returned. Specific observations can be protected from being deselected and

"kept" by passing a suitable argument to keep.these. Logical and integer vectors work as indexes to rows in data, while a values in a character vector are compared to the character values mapped to the label aesthetic. A function passed as argument to keep.these will receive as argument the values in the variable mapped to label and should return a character, logical or numeric vector as described above. If no variable has been mapped to label, row names are used in its place.

How many rows are retained in addition to those in keep.these is controlled with arguments passed to keep.number and keep.fraction. keep.number sets the maximum number of observations selected, whenever keep.fraction results in fewer observations selected, it is obeyed. If 'xintercept' is a finite value within the *x* range of the data and pool.along is passed "none" the data as are split into two groups and keep.number and keep.fraction are applied separately to each tail with density still computed jointly from all observations. If the length of keep.number and keep.fraction is one, this value is used for both tails, if their length is two, the first value is use for the left tail and the second value for the right tail.

Computation of density and of the default bandwidth require at least two observations with different values. If data do not fulfill this condition, they are kept only if keep.fraction = 1. This is correct behavior for a single observation, but can be surprising in the case of multiple observations.

Parameters keep.these and exclude.these make it possible to force inclusion or exclusion of observations after the density is computed. In case of conflict, exclude.these overrides keep.these.

Value

A plot layer instance. Using as output data a subset of the rows in input data retained based on a 1D filtering criterion.

Note

Which points are kept and which not depends on how dense and flexible is the density curve estimate. This depends on the values passed as arguments to parameters n, bw and kernel. It is also important to be aware that both geom_text() and geom_text_repel() can avoid over plotting by discarding labels at the plot rendering stage, i.e., what is plotted may differ from what is returned by this statistic.

See Also

density used internally.

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

Examples

```
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")

# looking under the hood with gginnards::geom_debug_group()
gginnards.installed <- requireNamespace("gginnards", quietly = TRUE)
if (gginnards.installed) {
  library(gginnards)

  ggplot(data = d, aes(x, y, label = lab, colour = group)) +
    stat_dens1d_filter(geom = "debug_group")
}

if (gginnards.installed) {
  ggplot(data = d, aes(x, y, label = lab, colour = group)) +
    stat_dens1d_filter(geom = "debug_group", return.density = TRUE)
}
```

`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` to restrict labeling to the low density tails of a distribution. By default the data are handled all together, but it is also possible to control labeling separately in each tail.

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,
  keep.these = FALSE,
  exclude.these = FALSE,
  these.target = "label",
  pool.along = c("x", "none"),
  xintercept = 0,
  invert.selection = FALSE,
  bw = "SJ",
  kernel = "gaussian",
  adjust = 1,
  n = 512,
  orientation = c("x", "y"),
  label.fill = "",
  return.density = FALSE,
  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 vector of length 1 or 2 [0..1]. The fraction of the observations (or rows) in data to be retained.
keep.number	integer vector of length 1 or 2. 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.
keep.these, exclude.these	character vector, integer vector, logical vector or function that takes one or more variables in data selected by these.target. Negative integers behave as in R's extraction methods. The rows from data indicated by keep.these and exclude.these are kept or excluded irrespective of the local density.

these.target	character, numeric or logical selecting one or more column(s) of data. If TRUE the whole data object is passed.
pool.along	character, one of "none" or "x", indicating if selection should be done pooling the observations along the *x* aesthetic, or separately on either side of xintercept.
xintercept	numeric The split point for the data filtering.
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.
return.density	logical vector of lenght 1. If TRUE add columns "density" and "keep.obs" to the returned data frame.
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 geometries 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, "". Function stat_dens1d_filter cannot be used with the repulsive geoms from 'ggrepel' because it drops the observations.

stat_dens1d_labels() can be useful also in other situations, as the substitution character string can be set by the user by passing an argument to label.fill. If this argument is NULL the unselected rows are filtered out.

The local density of observations along *x* or *y* is computed with function density and used to select observations, passing to the geom all the rows in its data input but with with the text of labels replaced in those "not kept". The default is to select observations in sparse regions of the plot, but the selection can be inverted so that only observations in the densest regions are returned. Specific observations can be protected from having the label replaced by passing a suitable argument to

keep.these. Logical and integer vectors function as indexes to rows in data, while a character vector is compared to values in the variable mapped to the label aesthetic. A function passed as argument to keep.these will receive as argument the values in the variable mapped to label and should return a character, logical or numeric vector as described above.

How many labels are retained intact in addition to those in keep.these is controlled with arguments passed to keep.number and keep.fraction. keep.number sets the maximum number of observations selected, whenever keep.fraction results in fewer observations selected, it is obeyed. If xintercept is a finite value within the *x* range of the data and pool.along is passed "none" the data are split into two groups and keep.number and keep.fraction are applied separately to each tail with density still computed jointly from all observations. If the length of keep.number and keep.fraction is one, half this value is used each tail, if their length is two, the first value is use for the left tail and the second value for the right tail (or if using orientation = "y" the lower and upper tails, respectively).

Computation of density and of the default bandwidth require at least two observations with different values. If data do not fulfill this condition, they are kept only if keep.fraction = 1. This is correct behavior for a single observation, but can be surprising in the case of multiple observations.

Parameters keep.these and exclude.these make it possible to force inclusion or exclusion of labels after the density is computed. In case of conflict, exclude.these overrides keep.these.

Value

A plot layer instance. Using as output data the input data after value substitution based on a 1D the filtering criterion.

Note

Which points are kept and which not depends on how dense and flexible is the density curve estimate. This depends on the values passed as arguments to parameters n, bw and kernel. It is also important to be aware that both geom_text() and geom_text_repel() can avoid overplotting by discarding labels at the plot rendering stage, i.e., what is plotted may differ from what is returned by this statistic.

See Also

density used internally.

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

Examples

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

# Make random data.
set.seed(1005)
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()

ggrepel.installed <- requireNamespace("ggrepel", quietly = TRUE)
if (ggrepel.installed) {
  library(ggrepel)

# 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")

  ggplot(data = d, aes(x, y)) +
    geom_point() +
    stat_dens1d_labels(geom = "text_repel", pool.along = "none")

  ggplot(data = d, aes(x, y)) +
    geom_point() +
    stat_dens1d_labels(geom = "text_repel",
                       keep.number = c(0, 10), pool.along = "none")

  ggplot(data = d, aes(x, y)) +
    geom_point() +
    stat_dens1d_labels(geom = "text_repel",
                       keep.fraction = c(0, 0.2), pool.along = "none")

# 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)

  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)
}

# Using geom_debug_group() we can see that all 100 rows in \code{d} are
# returned.
gginnards.installed <- requireNamespace("gginnards", quietly = TRUE)
if (gginnards.installed) {
  library(gginnards)

  ggplot(data = d, aes(x, y, label = lab)) +
    geom_point() +
    stat_dens1d_labels(geom = "debug_group")
}
if (gginnards.installed) {
  ggplot(data = d, aes(x, y, label = lab)) +
    geom_point() +
    stat_dens1d_labels(geom = "debug_group", return.density = TRUE)
}
```

`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. If there is no mapping to `label` in `data`, the mapping is silently set to `rownames(data)`.

Usage

```
stat_dens2d_filter(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  keep.fraction = 0.1,
```

```
  keep.number = Inf,
  keep.sparse = TRUE,
  keep.these = FALSE,
  exclude.these = FALSE,
  these.target = "label",
  pool.along = c("xy", "x", "y", "none"),
  xintercept = 0,
  yintercept = 0,
  invert.selection = FALSE,
  na.rm = TRUE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  h = NULL,
  n = NULL,
  return.density = FALSE
)

stat_dens2d_filter_g(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  keep.fraction = 0.1,
  keep.number = Inf,
  keep.sparse = TRUE,
  keep.these = FALSE,
  exclude.these = FALSE,
  these.target = "label",
  pool.along = c("xy", "x", "y", "none"),
  xintercept = 0,
  yintercept = 0,
  invert.selection = FALSE,
  na.rm = TRUE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  h = NULL,
  n = NULL,
  return.density = FALSE
)
```

Arguments

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

`...` 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.

`keep.these`, `exclude.these` character vector, integer vector, logical vector or function that takes one or more variables in data selected by `these.target`. Negative integers behave as in R's extraction methods. The rows from `data` indicated by `keep.these` and `exclude.these` are kept or excluded irrespective of the local density.

`these.target` character, numeric or logical selecting one or more column(s) of `data`. If `TRUE` the whole `data` object is passed.

`pool.along` character, one of `"none"`, `"x"`, `"y"`, or `"xy"` indicating if selection should be done pooling the observations along the *x*, `y`, both axes or none based on quadrants given by `xintercept` and `yintercept`.

`xintercept`, `yintercept` numeric The center point of the quadrants.

`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

`return.density` logical vector of lenght 1. If `TRUE` add columns `"density"` and `"keep.obs"` to the returned data frame.

Details

The local density of observations in 2D (*x* and *y*) is computed with function `kde2d` and used to select observations, passing to the geom a subset of the rows in its `data` input. The default is to select observations in sparse regions of the plot, but the selection can be inverted so that only observations in the densest regions are returned. Specific observations can be protected from being deselected and "kept" by passing a suitable argument to `keep.these`. Logical and integer vectors work as indexes to rows in `data`, while a character vector values are compared to the character values mapped to the `label` aesthetic. A function passed as argument to keep.these will receive

as argument the values in the variable mapped to `label` and should return a character, logical or numeric vector as described above. If no variable has been mapped to `label`, row names are used in its place.

How many rows are retained in addition to those in `keep.these` is controlled with arguments passed to `keep.number` and `keep.fraction`. `keep.number` sets the maximum number of observations selected, whenever `keep.fraction` results in fewer observations selected, it is obeyed.

Computation of density and of the default bandwidth require at least two observations with different values. If data do not fulfill this condition, they are kept only if `keep.fraction = 1`. This is correct behavior for a single observation, but can be surprising in the case of multiple observations.

Parameters `keep.these` and `exclude.these` make it possible to force inclusion or exclusion of observations after the density is computed. In case of conflict, `exclude.these` overrides `keep.these`.

Value

A plot layer instance. Using as output `data` a subset of the rows in input `data` retained based on a 2D-density-based filtering criterion.

Note

Which points are kept and which not depends on how dense a grid is used and how flexible the density surface estimate is. This depends on the values passed as arguments to parameters `n`, `bw` and `kernel`. It is also important to be aware that both `geom_text()` and `geom_text_repel()` can avoid overplotting by discarding labels at the plot rendering stage, i.e., what is plotted may differ from what is returned by this statistic.

See Also

`stat_dens2d_labels` and `kde2d` used internally. Parameters `n`, `h` in these statistics correspond to the parameters with the same name in this imported function. Limits are set to the limits of the plot scales.

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

Examples

```
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")

ggplot(data = d, aes(x, y, label = lab, colour = group)) +
  geom_point() +
  stat_dens2d_filter(geom = "text",
                     keep.these = function(x) {grepl("^u", x)})

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

```
  geom_point() +
  stat_dens2d_filter(geom = "text",
                     keep.these = function(x) {grepl("^u", x)})

ggplot(data = d, aes(x, y, label = lab, colour = group)) +
  geom_point() +
  stat_dens2d_filter(geom = "text",
                     keep.these = 1:30)

# looking under the hood with gginnards::geom_debug_group()
gginnards.installed <- requireNamespace("gginnards", quietly = TRUE)
if (gginnards.installed) {
  library(gginnards)

  ggplot(data = d, aes(x, y, label = lab, colour = group)) +
    stat_dens2d_filter(geom = "debug_group")
}
if (gginnards.installed) {
  ggplot(data = d, aes(x, y, label = lab, colour = group)) +
    geom_point() +
    stat_dens2d_filter(geom = "debug_group", return.density = TRUE)
}
```

`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,
  keep.these = FALSE,
  exclude.these = FALSE,
  these.target = "label",
  pool.along = c("xy", "x", "y", "none"),
```

```
  xintercept = 0,
  yintercept = 0,
  invert.selection = FALSE,
  h = NULL,
  n = NULL,
  label.fill = "",
  return.density = FALSE,
  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.
keep.these, exclude.these	character vector, integer vector, logical vector or function that takes one or more variables in data selected by these.target. Negative integers behave as in R's extraction methods. The rows from data indicated by keep.these and exclude.these are kept or excluded irrespective of the local density.
these.target	character, numeric or logical selecting one or more column(s) of data. If TRUE the whole data object is passed.
pool.along	character, one of "none" or "x", indicating if selection should be done pooling the observations along the *x* aesthetic, or separately on either side of xintercept.
xintercept, yintercept	numeric The split points for the data filtering.
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, a function or NULL.

return.density	logical vector of lenght 1. If TRUE add columns "density" and "keep.obs" to the returned data frame.
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 geometries from package 'ggrepel'. To avoid text labels being plotted over unlabelled points all the rows in data need to be retained but labels replaced with the empty character string, "". Function stat_dens2d_filter cannot be used with the repulsive geoms from 'ggrepel' because it drops observations.

stat_dens2d_labels() can be useful also in other situations, as the substitution character string can be set by the user by passing an argument to label.fill. If this argument is NULL the unselected rows are filtered out identically as by stat_dens2d_filter.

The local density of observations in 2D (x and y) is computed with function kde2d and used to select observations, passing to the geom all the rows in its data input but with with the text of labels replaced in those "not kept". The default is to select observations in sparse regions of the plot, but the selection can be inverted so that only observations in the densest regions are returned. Specific observations can be protected from having the label replaced by passing a suitable argument to keep.these. Logical and integer vectors function as indexes to rows in data, while a character vector is compared to values in the variable mapped to the label aesthetic. A function passed as argument to keep.these will receive as its first argument the values in the variable mapped to label and should return a character, logical or numeric vector as described above.

How many labels are retained intact in addition to those in keep.these is controlled with arguments passed to keep.number and keep.fraction. keep.number sets the maximum number of observations selected, whenever keep.fraction results in fewer observations selected, it is obeyed.

Computation of density and of the default bandwidth require at least two observations with different values. If data do not fulfill this condition, they are kept only if keep.fraction = 1. This is correct behavior for a single observation, but can be surprising in the case of multiple observations.

Parameters keep.these and exclude.these make it possible to force inclusion or exclusion of observations after the density is computed. In case of conflict, exclude.these overrides keep.these.

Value

A plot layer instance. Using as output data the input data after value substitution based on a 2D the filtering criterion.

Note

Which points are kept and which not depends on how dense a grid is used and how flexible the density surface estimate is. This depends on the values passed as arguments to parameters n, bw and kernel. It is also important to be aware that both geom_text() and geom_text_repel() can

avoid overplotting by discarding labels at the plot rendering stage, i.e., what is plotted may differ from what is returned by this statistic.

See Also

stat_dens2d_filter and kde2d used internally. Parameters n, h in this statistic correspond to the parameters with the same name in this imported function. Limits are set to the limits of the plot scales.

Other statistics returning a subset of data: stat_dens1d_filter(), stat_dens1d_labels(), stat_dens2d_filter()

Examples

```
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()

ggplot(data = d, aes(x, y, label = lab)) +
  geom_point() +
  stat_dens2d_labels(keep.these = "zoujdg")

ggplot(data = d, aes(x, y, label = lab)) +
  geom_point() +
  stat_dens2d_labels(keep.these = function(x) {grepl("^z", x)})

ggplot(data = d, aes(x, y, label = lab)) +
  geom_point() +
  stat_dens2d_labels(geom = "text_s",
                     position = position_nudge_center(x = 0.1, y = 0.1,
                                                      center_x = mean,
                                                      center_y = mean),
                     vjust = "outward_mean", hjust = "outward_mean") +
  expand_limits(x = c(-4, 4.5))

ggrepel.installed <- requireNamespace("ggrepel", quietly = TRUE)
if (ggrepel.installed) {
  library(ggrepel)
```

```
  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 function
# 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)
}

# Using geom_debug_group() we can see that all 100 rows in \code{d} are
# returned.
gginnards.installed <- requireNamespace("gginnards", quietly = TRUE)
if (gginnards.installed) {
  library(gginnards)

  ggplot(data = d, aes(x, y, label = lab)) +
    geom_point() +
    stat_dens2d_labels(geom = "debug_group")
}
if (gginnards.installed) {
  ggplot(data = d, aes(x, y, label = lab)) +
    geom_point() +
    stat_dens2d_labels(geom = "debug_group", return.density = TRUE)
}
if (gginnards.installed) {
  ggplot(data = d, aes(x, y)) +
    geom_point() +
    stat_dens2d_labels(geom = "debug_group")
}
```

---

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. It can also apply user supplied functions to data columns. 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,
  tb.funs = list(),
  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.
tb.funs	named list of functions to be applied to data columns.
digits	integer indicating the number of significant digits to be retained in data. Use digits = Inf to skip.
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.

Details

One or more functions to be applied can be passed in a named list to parameter 'tb.funs'. Functions are matched by name to columns, after column selection and renaming have been applied.

Value

A plot layer instance. Using as output data a copy of the input data 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))

# apply functions to columns
ggplot(my.df, aes(x, y, label = tbs)) +
  stat_fmt_tb(tb.vars = c(value = 1, group = 2),
              tb.rows = 1:3,
              tb.funs = list(group = function(x) {sprintf("italic(%s)", x)},
                             value = function(x) {ifelse(x > 2, "bold(zz)", x)}),
              parse = TRUE) +
  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_functions` *Draw functions as curves*

Description

`stat_functions()` computes values from functions and returns new data containing numeric vectors for `x` and `y`. As function definitions are passed through `data` this statistic follows the grammar of graphics in its behaviour.

Usage

```
stat_functions(
  mapping = NULL,
  data = NULL,
  n = 101,
  geom = "line",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
```

```
  ...
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with aes or aes_. Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset. Useful if the function curve is to be overlaid on other layers.
n	integer Number of points to interpolate along the x axis.
geom	The geometric object to use display the data
position	The position adjustment to use 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 it 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.
...	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 plot values computed by functions. As it follows the grammar of graphics, grouping and facets are supported. In this it differs from geom_function which behaves like a plot annotation.

Aesthetics xmin and xmax should be mapped to numeric values defining the range of the vector to be created and passed as argument to the function to compute the y values, and returned as x in data. n is the length of this x vector.

Value

A plot layer instance.

Computed variables

Data frame with n rows or a multiple of this, one for each row in data.

x numeric vector

y numeric vactor

idx integer vector, with values corresponding to rows in the input data, i.e., for each function

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.

Examples

```
# one function

df1 <- data.frame(min = 0, max = pi, fun = I(list(sin)))

ggplot(df1, aes(xmin = min, xmax = max, y = fun)) +
  stat_functions()

ggplot(df1, aes(xmin = min, xmax = max, y = fun)) +
  stat_functions(geom = "point", n = 20)

# two functions

df2 <- data.frame(min = -pi, max = pi,
                  fun = I(list(sin, cos)), name = c("sin", "cos"))

# each function must be in a separate group for correct plotting of lines

ggplot(df2, aes(xmin = min, xmax = max, y = fun, group = after_stat(idx))) +
  stat_functions()

ggplot(df2, aes(xmin = min, xmax = max, y = fun, colour = name)) +
  stat_functions()

ggplot(df2, aes(xmin = min, xmax = max, y = fun)) +
  stat_functions() +
  facet_grid(~ name)

# two curves with same function

df3 <- data.frame(min = c(-pi, 0),
                  max = c(0,pi),
                  fun = I(list(sin, sin)),
                  name = c("negative", "positive"))

ggplot(df3, aes(xmin = min, xmax = max, y = fun, colour = name)) +
  stat_functions()

# We use geom_debug_group() to see the computed values

gginnards.installed <- requireNamespace("gginnards", quietly = TRUE)
if (gginnards.installed) {
  library(gginnards)

  ggplot(df1, aes(xmin = min, xmax = max, y = fun)) +
    stat_functions(geom = "debug_group")
}
```

`stat_panel_counts` *Number of observations in a plot panel*

Description

`stat_panel_counts()` counts the number of observations in each panel. `stat_group_counts()` counts the number of observations in each group. By default they add one or more text labels to the top right corner of each panel. Grouping is ignored by `stat_panel_counts()`. If no grouping exists, the two statistics behave similarly.

Usage

```
stat_panel_counts(
  mapping = NULL,
  data = NULL,
  geom = "text_npc",
  position = "identity",
  label.x = "right",
  label.y = "top",
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  ...
)

stat_group_counts(
  mapping = NULL,
  data = NULL,
  geom = "text_npc",
  position = "identity",
  label.x = "right",
  label.y = "top",
  hstep = 0,
  vstep = NULL,
  digits = 2,
  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. Rarely used, as you will not want to override the plot defaults.
`geom`	The geometric object to use display the data
`position`	The position adjustment to use on this layer
`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 it 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.
hstep, vstep	numeric in npc units, the horizontal and vertical step used between labels for different groups.
digits	integer Number of digits for fraction and percent labels.

Details

These statistics can be used to automatically count observations in each panel of a plot, and by default add these counts as text labels. These statistics, unlike `stat_quadrant_counts()` requires only one of *x* or *y* aesthetics and can be used together with statistics that have the same requirement, like `stat_density()`.

The default position of the label is in the top right corner. 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 to `label.x` and `label.y`, pass also explicitly `geom = "text"`, `geom = "label"` or some other geometry that use the *x* and/or *y* aesthetics. A vector with the same length as the number of panels in the figure can be used if needed.

Value

A plot layer instance. Using as output `data` the counts of observations in each plot panel or per group in each plot panel.

Computed variables

Data frame with one or more rows, one for each group of observations for which counts are counted in `data`.

x,npcx x value of label position in data- or npc units, respectively

y,npcy y value of label position in data- or npc units, respectively

count number of observations as an integer

count.label number of observations as character

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. `x` and `y` are included in the output only if mapped.

Note

If a factor is mapped to `x` or to `y` aesthetics each level of the factor constitutes a group, in this case the default positioning and geom using NPC pseudo aesthetics will have to be overriden by passing `geom = "text"` and data coordinates used. The default for factors may change in the future.

See Also

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

Examples

```
# generate artificial data with numeric x and y
set.seed(67821)
x <- 1:100
y <- rnorm(length(x), mean = 10)
group <- factor(rep(c("A", "B"), times = 50))
my.data <- data.frame(x, y, group)

# using automatically generated text labels

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

ggplot(my.data, aes(x, y, colour = group)) +
  geom_point() +
  stat_panel_counts()

ggplot(my.data, aes(x, y, colour = group)) +
  geom_point() +
  stat_group_counts()

ggplot(my.data, aes(x, y, colour = group)) +
  geom_point() +
  stat_group_counts(label.x = "left", hstep = 0.06, vstep = 0)

ggplot(my.data, aes(x, y, colour = group)) +
  geom_point() +
  stat_group_counts(aes(label = after_stat(pc.label)))

ggplot(my.data, aes(x, y, colour = group)) +
  geom_point() +
  stat_group_counts(aes(label = after_stat(pc.label)), digits = 3)

ggplot(my.data, aes(x, y, colour = group)) +
  geom_point() +
  stat_group_counts(aes(label = after_stat(fr.label)))

ggplot(my.data, aes(x, y, colour = group)) +
  geom_point() +
  stat_group_counts(aes(label = after_stat(dec.label)))
```

```
# one of x or y can be a factor
# label.x or label.y along the factor can be set to "factor" together
# with the use of geom_text()

ggplot(mpg,
       aes(factor(cyl), hwy)) +
  stat_boxplot() +
  stat_group_counts(geom = "text",
                    label.y = 10,
                    label.x = "factor") +
  stat_panel_counts()

# Numeric values can be used to build labels with alternative formats
# Here with sprintf(), but paste() and format() also work.

ggplot(my.data, aes(x, y)) +
 geom_point() +
 stat_panel_counts(aes(label = sprintf("%i observations",
                                       after_stat(count)))) +
 scale_y_continuous(expand = expansion(mult = c(0.05, 0.12)))

ggplot(mpg,
       aes(factor(cyl), hwy)) +
  stat_boxplot() +
  stat_group_counts(geom = "text",
                    aes(label = sprintf("(%i)", after_stat(count))),
                    label.y = 10,
                    label.x = "factor")

ggplot(mpg,
       aes(factor(cyl), hwy)) +
  stat_boxplot() +
  stat_group_counts(aes(label = sprintf("n[%i]~`=`~%i",
                                        after_stat(x), after_stat(count))),
                    parse = TRUE,
                    geom = "text",
                    label.y = 10,
                    label.x = "factor") +
  stat_panel_counts(aes(label = sprintf("sum(n[i])~`=`~%i",
                                        after_stat(count))),
                    parse = TRUE)

# label position

ggplot(my.data, aes(y)) +
  stat_panel_counts(label.x = "left") +
  stat_density(alpha = 0.5)

ggplot(my.data, aes(y, colour = group)) +
  stat_group_counts(label.y = "top") +
  stat_density(aes(fill = group), alpha = 0.3)

# The numeric value can be used as a label as is
```

```
ggplot(mpg,
       aes(factor(cyl), hwy)) +
  stat_boxplot() +
  stat_group_counts(geom = "text",
                    aes(label = after_stat(count)),
                    label.x = "factor",
                    label.y = 10) +
  annotate(geom = "text", x = 0.55, y = 10, label = "n[i]~`=`", parse = TRUE)

# We use geom_debug_group() to see the computed values

gginnards.installed <- requireNamespace("gginnards", quietly = TRUE)
if (gginnards.installed) {
  library(gginnards)

  ggplot(my.data, aes(x, y)) +
    geom_point() +
    stat_panel_counts(geom = "debug_group")
}

if (gginnards.installed) {
  ggplot(my.data, aes(x, y, colour = group)) +
    geom_point() +
    stat_group_counts(geom = "debug_group")
}
```

`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 = c("none", "x", "y", "xy"),
  xintercept = 0,
  yintercept = 0,
```

```
  label.x = NULL,
  label.y = NULL,
  digits = 2,
  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 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.
digits	integer Number of digits for fraction and percent 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.

Value

A plot layer instance. Using as output `data` the counts of observations per plot quadrant.

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 in the quadrant(s)

total number of onservations in data

count.label number of observations as character

pc.label percent of observations as character

fr.label fraction of observations as character

.

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: `geom_quadrant_lines()`, `stat_panel_counts()`

Examples

```
# generate artificial data
set.seed(4321)
x <- -50:50
y <- rnorm(length(x), mean = 0)
my.data <- data.frame(x, y)

# using automatically generated text labels, default origin at (0, 0)

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

```
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_quadrant_lines() +
  stat_quadrant_counts(aes(label = after_stat(pc.label)))

ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_quadrant_lines() +
  stat_quadrant_counts(aes(label = after_stat(fr.label)))

ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_quadrant_lines() +
  stat_quadrant_counts(aes(label = after_stat(dec.label)))

ggplot(my.data, aes(x, y)) +
 geom_point() +
  geom_quadrant_lines() +
  stat_quadrant_counts(aes(label = sprintf("%i observations", after_stat(count)))) +
  scale_y_continuous(expand = expansion(c(0.05, 0.15))) # reserve space

# user specified origin

ggplot(my.data, aes(x, y)) +
  geom_quadrant_lines(colour = "blue", xintercept = 10, yintercept = -1) +
  stat_quadrant_counts(colour = "blue", xintercept = 10, yintercept = -1) +
  geom_point() +
  scale_y_continuous(expand = expansion(mult = 0.15))

ggplot(my.data, aes(x, y)) +
  geom_quadrant_lines(colour = "blue", xintercept = 10, yintercept = -1) +
  stat_quadrant_counts(aes(label = after_stat(pc.label)),
                       colour = "blue", xintercept = 10, yintercept = -1) +
  geom_point() +
  scale_y_continuous(expand = expansion(mult = 0.15))

# more digits in labels

ggplot(my.data, aes(x, y)) +
  geom_quadrant_lines(colour = "blue", xintercept = 10, yintercept = -1) +
  stat_quadrant_counts(aes(label = after_stat(pc.label)), digits = 3,
                       colour = "blue", xintercept = 10, yintercept = -1) +
  geom_point() +
  scale_y_continuous(expand = expansion(mult = 0.15))

ggplot(my.data, aes(x, y)) +
  geom_quadrant_lines(colour = "blue", xintercept = 10, yintercept = -1) +
  stat_quadrant_counts(aes(label = after_stat(fr.label)),
                       colour = "blue", xintercept = 10, yintercept = -1) +
  geom_point() +
  scale_y_continuous(expand = expansion(mult = 0.15))
```

```
# grouped quadrants

ggplot(my.data, aes(x, y)) +
  geom_quadrant_lines(colour = "blue",
                      pool.along = "x") +
  stat_quadrant_counts(colour = "blue", label.x = "right",
                       pool.along = "x") +
  geom_point() +
  scale_y_continuous(expand = expansion(mult = 0.15))

# whole panel

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quadrant_counts(quadrants = 0, label.x = "left", label.y = "bottom") +
  scale_y_continuous(expand = expansion(mult = c(0.15, 0.05)))

# use a different geometry

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

# Numeric values can be used to build labels with alternative formats
# Here with sprintf(), but paste() and format() also work.

ggplot(my.data, aes(x, y)) +
  geom_quadrant_lines(colour = "blue") +
  stat_quadrant_counts(aes(label = sprintf("%i of %i genes",
                       after_stat(count), after_stat(total))),
                       colour = "blue") +
  geom_point() +
  scale_y_continuous(expand = expansion(mult = 0.15))

# We use geom_debug_group() to see the computed values

gginnards.installed <- requireNamespace("gginnards", quietly = TRUE)
if (gginnards.installed) {
  library(gginnards)

  ggplot(my.data, aes(x, y)) +
    geom_point() +
    stat_quadrant_counts(geom = "debug_group")
}

if (gginnards.installed) {
  ggplot(my.data, aes(x, y)) +
    geom_point() +
    stat_quadrant_counts(geom = "debug_group", xintercept = 50)
}
```

`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 = grid::unit(c(0.8, 0.6), "char"),
  text.alpha = NA,
  rules.alpha = NA,
  canvas.alpha = NA,
  ...
)

ttheme_gtminimal(
  base_size = 10,
  base_colour = "black",
  base_family = "",
  parse = FALSE,
  padding = grid::unit(c(0.5, 0.4), "char"),
  text.alpha = NA,
  rules.alpha = NA,
```

```
  canvas.alpha = NA,
  ...
)

ttheme_gtbw(
  base_size = 10,
  base_colour = "black",
  base_family = "",
  parse = FALSE,
  padding = grid::unit(c(1, 0.6), "char"),
  text.alpha = NA,
  rules.alpha = NA,
  canvas.alpha = NA,
  ...
)

ttheme_gtplain(
  base_size = 10,
  base_colour = "black",
  base_family = "",
  parse = FALSE,
  padding = grid::unit(c(0.8, 0.6), "char"),
  text.alpha = NA,
  rules.alpha = NA,
  canvas.alpha = NA,
  ...
)

ttheme_gtdark(
  base_size = 10,
  base_colour = "grey90",
  base_family = "",
  parse = FALSE,
  padding = grid::unit(c(0.8, 0.6), "char"),
  text.alpha = NA,
  rules.alpha = NA,
  canvas.alpha = NA,
  ...
)

ttheme_gtlight(
  base_size = 10,
  base_colour = "grey10",
  base_family = "",
  parse = FALSE,
  padding = grid::unit(c(0.8, 0.6), "char"),
  text.alpha = NA,
  rules.alpha = NA,
```

```
  canvas.alpha = NA,
  ...
)

ttheme_gtsimple(
  base_size = 10,
  base_colour = "grey10",
  base_family = "",
  parse = FALSE,
  padding = grid::unit(c(0.5, 0.4), "char"),
  text.alpha = NA,
  rules.alpha = NA,
  canvas.alpha = NA,
  ...
)

ttheme_gtstripes(
  base_size = 10,
  base_colour = "grey10",
  base_family = "",
  parse = FALSE,
  padding = grid::unit(c(0.8, 0.6), "char"),
  text.alpha = NA,
  rules.alpha = NA,
  canvas.alpha = NA,
  ...
)
```

Arguments

base_size	numeric, default font size of text in table.
base_colour	default font colour for text in table.
base_family	default font family for text in table.
parse	logical, behaviour for parsing text as plotmath.
padding	length-2 unit vector specifying the horizontal and vertical padding of text within each cell.
text.alpha, canvas.alpha, rules.alpha	numeric in [0..1] Transparency applied to table base_colour, to table body background fill and rules colour, respectively.
...	further arguments to control the gtable.

Details

These wrapper functions are table theme (ttheme) constructors making it easier to change the style of tables created with tableGrob. When passed as argument to geom_table the table theme's base_colour, base_family, base_colour and base_size function as defaults for the text in the body of the table. They are overriden if the corresponding text-related aesthetics are mapped or

set to a constant through the usual 'ggplot2' mechanisms. On the other hand the properties of the background fill, rules and column and row headings can be set only through the theme. The `ttheme` constructors defined in 'ggpp' have formal parameters for `alpha` transparency of the text, background and rules. Transparency is useful as plot insets can accidentally overlap observations hiding them from view depending on the stacking order of plot layers.

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

Value

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

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()

# Base colour of theme overridden 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()

# Base colour of theme overridden 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)

# Base colour of theme overridden 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")

# Base colour of theme overridden 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()

# Transparency of table background fill and grid lines colour
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df, aes(x = 1.5, y = y, label = tb),
            table.theme = ttheme_gtplain(canvas.alpha = 0.5,
                                         rules.alpha = 0.2)) +
  theme_classic()

# Transparency of table background fill and grid lines colour
# and table text base colour: black with 50% transparency
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
```

```
  geom_point() +
  geom_table(data = df, aes(x = 1.5, y = y, label = tb),
             table.theme = ttheme_gtplain(text.alpha = 0.5)) +
  theme_classic()
```

`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)

set_ttheme(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` *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

References

Rai, Neha; O'Hara, Andrew; Farkas, Daniel; Safronov, Omid; Ratanasopa, Khuanpiroon; Wang, Fang; Lindfors, Anders V.; Jenkins, Gareth I.; Lehto, Tarja; Salojärvi, Jarkko; Brosché, Mikael; Strid. Åke; Aphalo, Pedro José; Morales, Luis Orlando (2020) The photoreceptor UVR8 mediates the perception of both UV-B and UV-A wavelengths up to 350 nm of sunlight with responsivity moderated by cryptochromes. *Plant, Cell & Environment*, 43:1513-1527.

See Also

Other Transcriptomics data: `quadrant_example.df`

Examples

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

`weather_18_june_2019.df` *Weather data*

Description

A data set containing weather data measured in Viikki, Helsinki, Finland. Values for all variables are means of 12 readings at 5 seconds intervals. Sun angles were computed with R package 'photobiology'.

Usage

```
weather_18_june_2019.df
```

Format

A tibble with 18 columns and 1440 rows.

Details

The variables are as follows:

- time (yyyy-mm-dd hh:mm:ss)
- PAR_umol (umol m-2 s-1)
- PAR_diff_fr (/1)
- global_watt (W m-2)
- day_of_year
- month_of_year
- month_name
- calendar_year
- solar_time (h)
- sun_elevation (degrees above horizon)
- sun_azimuth (degrees)
- was_sunny (T/F)
- wind_speed (m s-1)
- wind_direction (degrees)
- air_temperature_C (C)
- air_RH (
- air_DP (C)
- air_pressure

References

P. J. Aphalo, unpublished data.

Examples

```
names(weather_18_june_2019.df)
head(weather_18_june_2019.df)
nrow(weather_18_june_2019.df)
```

`wrap_labels` *Wrap character strings in a vector*

Description

Wrap the members of a character vector to a given maximum width by inserting new line characters at word boundaries.

Usage

```
wrap_labels(x, width = 20, indent = 0, new.line = "\n")
```

Arguments

`x`	character vector, or an object which can be converted to a character vector by `as.character`.
`width`	a positive integer giving the target column for wrapping lines in the output.
`indent`	a positive or negative integer giving the indentation of the first line in a member character string.
`new.line`	character sting; use `" "` for HTML encoded strings.

Details

Function `wrap_labels()` is a wrapper on `link{strwrap}` that returns a vector of character strings instead of a list of vectors. In addition to wrapping, indentation is supported. Wrapping is always at white space, so `width = 0` wraps word by word.

Because the returned value is a character vector of the same length as the input, this function can be used within a call to `aes()` when mapping a character vector to the `label` aesthetic, as long as the character strings will not be parsed into R expressions. It can be also used to wrap the strings in a variable stored in a data frame.

Value

A character vector of the same length as `x`, with new line characters inserted to wrap text lines longer than `width`. Names in `x` are preserved in the returned value, no names are added if none are present in `x`.

Examples

```
my.text <- c(A = "This is the first string",
             B = "This is the second string, which is longer")

wrap_labels(my.text, width = 20)
wrap_labels(unname(my.text), width = 20)

cat(wrap_labels(my.text, width = 20), sep = "\n--\n")
cat(wrap_labels(my.text, width = 20, indent = 2), sep = "\n--\n")
cat(wrap_labels(my.text, width = 20, indent = -2), sep = "\n--\n")
```

Index