Package 'ggpp'

December 5, 2022

Type Package

Title Grammar Extensions to 'ggplot2'

Version 0.5.0

Date 2022-12-05

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)

LazyLoad TRUE

ByteCompile TRUE

Depends R (>= 4.0.0), ggplot2 (>= 3.3.6)

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

Suggests knitr (>= 1.29), rmarkdown (>= 2.3), ggrepel (>= 0.9.2),
gginnards(>= 0.1.0), magick (>= 2.6.0)

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

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

Encoding UTF-8

RoxygenNote 7.2.2

VignetteBuilder knitr

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

R topics documented:

| 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]

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)
```

---

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 chan-
nels 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 in [0..1] indicating the switch point between dark and light
                background.

dark.color, light.color
                A color definition or color name to return for dark and light objects (orveplotted
                on light and dark backgrounds, respectively).

Examples

```
dark_or_light("yellow")
dark_or_light("darkblue")
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.

Usage

```
geom_grob(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  nudge_x = 0,
  nudge_y = 0,
  default.colour = "black",
  colour.target = "segment",
  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 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	A colour definition to use for elements not targeted by the colour aesthetic.
colour.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 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.

This geom_grob and geom_grob_npc require the use tibbles as argument for data, as the grobs should be stored as a list of graphics objects ("grob") to be mapped to the label aesthetic.

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")

# 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)
```

---

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,
  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.
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

This geom is identical to 'ggplot2' [geom_text]() except that it interprets npcx and npcy positions in npc units. It translates npcx and npcy coordinates using a pseudo-aesthetic with a fixed scale. The generation of grobs is done with calls to functions imported from package 'ggplot2'.

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](#)

Examples

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

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

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

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

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

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

```
geom_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 = "black",
  colour.target = "all",
  default.alpha = 1,
  alpha.target = "box.fill",
  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,
  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 = "black",
```

```
    colour.target = "all",
    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,
    check_overlap = FALSE,
    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	A colour definition to use for elements not targeted by the colour aesthetic.
colour.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"`.

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
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.

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 our geoms attempts to enforce this.

Value

A plot layer instance.

Under development!

These two geometries are still under development and their user interface subject to change. In 'ggpp' (== 0.5.0) support for aesthetics related to segments was removed, and replaced with parameters and a new mechanism for targeting a the usual aesthetics to text, border, and segment was added.

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.

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]().

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)
p <- ggplot(my.cars, aes(wt, mpg, label = name)) +
       geom_point(color = "red")

# Use 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"))) +
  expand_limits(x = 6.2)
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(hjust = "left", nudge_x = 0.12) +
  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(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 = 0.4) +
  expand_limits(x = 6.2)
p +
  geom_label_s(nudge_x = 0.12, 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)

  # 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)

  # 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"),
                show.legend = FALSE) +
    scale_colour_discrete(l = 40) + # luminance, make colours darker
    expand_limits(y = 27)

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

  p +
    geom_label_s(aes(colour = factor(cyl)),
                 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 = "text",
                 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 = 0.6,
               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 = "black",
      colour.target = "segment",
      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_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 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.
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	A colour definition to use for elements not targeted by the colour aesthetic.
colour.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.

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. In contrast to [position_nudge](#), [position_nudge_center](#) and all other position functions defined in packages 'ggpp' and 'ggrepel' keep the original coordinates thus allowing the plotting of connecting segments and arrows.

This geom works only with tibbles as data, as its expects a list of ggplot objects (″gg″ class) to be mapped to the label aesthetic.

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.

In the case of geom_plot_npc(), npcx and npcy aesthetics determine the position of the inset plot. Justification as described above for .

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()

df <- tibble(x = 0.01,
             y = 0.01,
             plot = list(p +
                           coord_cartesian(xlim = c(3, 4),
                                           ylim = c(13, 16)) +
                           labs(x = NULL, y = NULL) +
                           theme_bw(10)))
p +
  expand_limits(x = 0, y = 0) +
  geom_plot_npc(data = df,
                aes(npcx = x, npcy = y, label = plot))

p +
  expand_limits(x = 0, y = 0) +
  geom_plot_npc(data = df,
                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 dis-
placed observations to their current position rendered as points or graphic symbols. The most com-
mon 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,
  arrow = grid::arrow(length = unit(1/3, "lines")),
  default.colour = "black",
  colour.target = "point",
  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,
  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.
arrow	specification for arrow heads, as created by [arrow]()

`default.colour`	A colour definition to use for elements not targeted by the colour aesthetic.
`colour.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.

`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()

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

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

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

---

geom_quadrant_lines   *Reference lines: horizontal plus vertical, and quadrants*

Description

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

Usage

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

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

Arguments

mapping	The aesthetic mapping, usually constructed with [aes](#). Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific data set - only needed if you want to override the plot defaults.
stat	The statistic object to use display the data
position	The position adjustment to use for overlapping points on this layer
pool.along	character, one of "none", "x" or "y", indicating which quadrants to pool to calculate counts by pair of quadrants.

```
xintercept, yintercept
                numeric vectors the coordinates of the origin of the quadrants.
```

na.rm   a logical indicating whether NA values should be stripped before the computa-
                tion proceeds.

show.legend   logical. Should this layer be included in the legends? NA, the default, includes if
                any aesthetics are mapped. FALSE never includes, and TRUE always includes.

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

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

Details

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

---

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 = "black",
  colour.target = "segment",
  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,
    table.theme = NULL,
    table.rownames = FALSE,
    table.colnames = TRUE,
    table.hjust = 0.5,
    parse = FALSE,
    na.rm = FALSE,
    show.legend = FALSE,
    inherit.aes = FALSE
)

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

Arguments

mapping	The aesthetic mapping, usually constructed with [aes](). Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific data set - only needed if you want to override the plot defaults.
stat	The statistical transformation to use on the data for this layer, as a string.
position	Position adjustment, either as a string, or the result of a
...	other arguments passed on to [layer](). This can include aesthetics whose values you want to set, not map. See [layer]() for more details.
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	A colour definition to use for elements not targeted by the colour aesthetic.
colour.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](#)

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

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

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

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

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

show.legend   logical. Should this layer be included in the legends? NA, the default, includes if
                  any aesthetics are mapped. FALSE never includes, and TRUE always includes.

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

Details

You can modify the size of inset tables 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 de-
fault 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.

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. In contrast to [position_nudge](#),
[position_nudge_center](#) and all other position functions defined in packages 'ggpp' and 'ggre-
pel' keep the original coordinates thus allowing the plotting of connecting segments and arrows.

This geom works only with tibbles as data, as its expects a list of data frames (or tibbles) to be
mapped to the label aesthetic. A table is built with function gridExtra::gtable for each data
frame in the list, and formatted according to a table theme or ttheme. The character strings in the
data frame can be parsed into R expressions so the inset tables can include maths.

If the argument passed to table.theme is a constructor function (passing its name without paren-
thesis), 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 will be used as is, i.e., mappings to aesthetics such as colour are ignored if

present. By default the constructor `ttheme_gtdefault` is used and `colour` and `fill`, are mapped to NA. Mapping these aesthetics to NA triggers the use of the default `base_colour` of the `ttheme`. As the table is built with function `gridExtra::gtable()`, for formatting details, please, consult [tableGrob](#).

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.

In the case of `geom_table_npc`, `npcx` and `npcy` aesthetics determine the position of the inset table. Justification as described above for .

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)

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

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

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

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

# settings aesthetics to constants
ggplot(mtcars,
       aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
  geom_table(data = df,
```

```
              aes(x = x, y = y, label = tb),
              color = "red", fill = "#FFCCCC",
              family = "serif", size = 5,
              angle = 90, vjust = 0)

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

df2 <- tibble(x = 5.45,
              y = c(34, 29, 24),
              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))

# 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()

# 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 "dou-
ble 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

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

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 to text labels linked to observations plotted without dodge. 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_dodgenudge(
  width = 1,
  preserve = c("total", "single"),
  x = 0,
  y = 0,
  direction = "none",
  kept.origin = "dodged"
)

position_dodge_keep(width = 1, preserve = c("total", "single"))

position_dodge2_keep(width = 1, preserve = c("total", "single"))

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

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?.

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',
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" 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.
reverse	If TRUE, will reverse the default stacking order. This is useful if you're rotating both the plot and legend.

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 posible 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 jittering is computed twice. Jitter should be identical with the same arguments as position_dodgenudge() as this last function simply call the same code from package 'ggplot2'.

The second use is to dodge labels to be connected to elements that have not been jittered. 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_jitternudge](), [position_nudge_center](), [position_nudge_line](), [position_nudge_to](), [position_stacknudge]()

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_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 = "as.is",
```

```
  nudge.from = "original",
  kept.origin = "jittered"
)
```

```
position_jitter_keep(width = NULL, height = NULL, seed = NA)
```

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`,
`direction`	One of `"as.is"`, code"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_nudge_center(), position_nudge_line(),
position_nudge_to(), position_stacknudge()

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 = "original"
)
```

```
position_nudge_centre(
  x = 0,
  y = 0,
  center_x = NULL,
  center_y = NULL,
  direction = NULL,
  obey_grouping = NULL,
  kept.origin = "original"
)
```

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

Arguments

x, y   Amount of vertical and horizontal distance to move. A numeric vector of length
                1, or of the same length as rows there are in data,

center_x, center_y

                The coordinates of the virtual origin out from which nudging radiates or splits
                in opposite directions. A numeric vector of length 1 or of the same length as
                rows there are in data, or a function returning either of these vectors computed
                from the variables in data mapped to x or y, respectively.

direction   One of "none", "radial", or "split". A value of "none" replicates the behav-
                ior 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 emphcenter point.

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

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

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)) +
  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))

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

ggplot(df, aes(x, z, color = group)) +
  geom_point() +
  geom_line(color = "black", linetype = "dotted") +
  geom_text_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_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 com-
pared 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* val-
ues. 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 = NULL,
  line_nudge = 1,
  kept.origin = "original"
)
```

Arguments

x, y	Amount of vertical and horizontal distance to move. A numeric vector of length 1, or of the same length as rows there are in data.
xy_relative	Nudge relative to *x* and *y* data expanse, ignored unless x and y are both 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 "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_jitternudge(), position_nudge_center(), position_nudge_to(), position_stacknudge()

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

Usage

```
position_nudge_to(x = NULL, y = NULL, kept.origin = "original")
```

Arguments

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

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

Details

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

Value

A "Position" object.

See Also

position_nudge, position_nudge_repel.

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

Examples

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

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

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

---

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 = "none",
  kept.origin = "stacked"
)

position_fillnudge(
  vjust = 1,
```

```
  reverse = FALSE,
  x = 0,
  y = 0,
  direction = "none",
  kept.origin = "stacked"
)

position_stack_keep(vjust = 1, reverse = FALSE)

position_fill_keep(vjust = 1, reverse = FALSE)
```

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,
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

This position function is 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_nudge_keep() with exactly the same signature and behaviour as [position_nudge]() provides an easier to remember name when the desire is only to have access to both the original and nudged coordinates.

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_jitternudge(), position_nudge_center(), position_nudge_line(), position_nudge_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")
```

---

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", size = 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()

# 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")

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

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

  ggplot(my.df, aes(x = X, y = Y, colour = category)) +
    geom_point() +
    stat_apply_group(geom = "debug",
                     .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.

Usage

```
stat_dens1d_filter(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  keep.fraction = 0.1,
  keep.number = Inf,
  keep.sparse = TRUE,
  invert.selection = FALSE,
  bw = "SJ",
  kernel = "gaussian",
  adjust = 1,
  n = 512,
  orientation = "x",
  na.rm = TRUE,
  show.legend = FALSE,
  inherit.aes = TRUE
)

stat_dens1d_filter_g(
```

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

Arguments

mapping	The aesthetic mapping, usually constructed with [aes](#) or [aes_](#). Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to [layer](#). This can include aesthetics whose values you want to set, not map. See [layer](#) for more details.
keep.fraction	numeric [0..1]. The fraction of the observations (or rows) in data to be retained.
keep.number	integer Set the maximum number of observations to retain, effective only if obeying keep.fraction would result in a larger number.
keep.sparse	logical If TRUE, the default, observations from the more sparse regions are retained, if FALSE those from the densest regions.
invert.selection	logical If TRUE, the complement of the selected rows are returned.
bw	numeric or character The smoothing bandwidth to be used. If numeric, the standard deviation of the smoothing kernel. If character, a rule to choose the bandwidth, as listed in [bw.nrd](#).
kernel	character See [density](#) for details.
adjust	numeric A multiplicative bandwidth adjustment. This makes it possible to adjust the bandwidth while still using the a bandwidth estimator through an argument passed to bw. The larger the value passed to adjust the stronger the smoothing, hence decreasing sensitivity to local changes in density.
n	numeric Number of equally spaced points at which the density is to be estimated for applying the cut point. See [density](#) for details.

orientation	character The aesthetic along which density is computed. Given explicitly by setting orientation to either "x" or "y".
na.rm	a logical value indicating whether NA values should be stripped before the computation proceeds.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders.

Value

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

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")

# repulsive labels with ggrepel::geom_text_repel()
ggrepel.installed <- requireNamespace("ggrepel", quietly = TRUE)
if (ggrepel.installed) {
  library(ggrepel)

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

---

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

Description

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

Usage

```
stat_dens1d_labels(
  mapping = NULL,
  data = NULL,
  geom = "text",
  position = "identity",
  ...,
  keep.fraction = 0.1,
  keep.number = Inf,
  keep.sparse = TRUE,
  invert.selection = FALSE,
  bw = "SJ",
  kernel = "gaussian",
  adjust = 1,
  n = 512,
  orientation = "x",
  label.fill = "",
  na.rm = TRUE,
  show.legend = FALSE,
  inherit.aes = TRUE
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with [aes](aes) or [aes_](aes_). Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data.
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
keep.fraction	numeric [0..1]. The fraction of the observations (or rows) in data to be retained.
keep.number	integer Set the maximum number of observations to retain, effective only if obeying keep.fraction would result in a larger number.
keep.sparse	logical If TRUE, the default, observations from the more sparse regions are retained, if FALSE those from the densest regions.
invert.selection	
	logical If TRUE, the complement of the selected rows are returned.
bw	numeric or character The smoothing bandwidth to be used. If numeric, the standard deviation of the smoothing kernel. If character, a rule to choose the bandwidth, as listed in [bw.nrd](bw.nrd).
kernel	character See density for details.
adjust	numeric A multiplicative bandwidth adjustment. This makes it possible to adjust the bandwidth while still using the a bandwidth estimator through an argument passed to bw. The larger the value passed to adjust the stronger the smoothing, hence decreasing sensitivity to local changes in density.
n	numeric Number of equally spaced points at which the density is to be estimated for applying the cut point. See density for details.
orientation	character The aesthetic along which density is computed. Given explicitly by setting orientation to either "x" or "y".
label.fill	character vector of length 1 or a function.
na.rm	a logical value indicating whether NA values should be stripped before the computation proceeds.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders.

Details

stat_dens1d_labels() is designed to work together with statistics from package 'ggrepel'. To avoid text labels being plotted over unlabelled points the corresponding rows in data need to be retained but labels replaced with the empty character string, `""`. This makes stat_dens1d_filter unsuitable for the task. Non-the-less stat_dens1d_labels() could be useful in some other cases, as the substitution character string can be set by the user.

Value

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

See Also

[density](#) used internally.

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

Examples

```
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")

# 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() we can see that all 100 rows in \code{d} are
# returned. But only those labelled in the previous example still contain
# the original labels.

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")
}
```

---

stat_dens2d_filter   *Filter observations by local 2D density*

Description

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

Usage

```
stat_dens2d_filter(
```

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

stat_dens2d_filter_g(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  keep.fraction = 0.1,
  keep.number = Inf,
  keep.sparse = TRUE,
  invert.selection = FALSE,
  na.rm = TRUE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  h = NULL,
  n = NULL,
  ...
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with [aes](#) or [aes_](#). Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data.
position	The position adjustment to use for overlapping points on this layer
keep.fraction	numeric [0..1]. The fraction of the observations (or rows) in data to be retained.
keep.number	integer Set the maximum number of observations to retain, effective only if obeying keep.fraction would result in a larger number.
keep.sparse	logical If TRUE, the default, observations from the more sparse regions are retained, if FALSE those from the densest regions.
invert.selection	
	logical If TRUE, the complement of the selected rows are returned.

na.rm	a logical value indicating whether NA values should be stripped before the computation proceeds.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders.
h	vector of bandwidths for x and y directions. Defaults to normal reference bandwidth (see bandwidth.nrd). A scalar value will be taken to apply to both directions.
n	Number of grid points in each direction. Can be scalar or a length-2 integer vector
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.

Value

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

See Also

kde2d used internally.

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

Examples

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

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

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

# filter observations not in the sparsest regions
ggplot(data = d, aes(x, y)) +
```

```
  geom_point() +
  stat_dens2d_filter(colour = "blue", invert.selection = TRUE)

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

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

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

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

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

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

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

# repulsive labels with ggrepel::geom_text_repel()
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_filter(geom = "text_repel")
}
```

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

Description

stat_dens2d_labels() Sets values mapped to the label aesthetic to "" or a user provided character string based on the local density in regions of a plot panel. Its main use is together with repulsive geoms from package [ggrepel](). If there is no mapping to label in data, the mapping is set to rownames(data), with a message.

Usage

```
stat_dens2d_labels(
  mapping = NULL,
  data = NULL,
  geom = "text",
  position = "identity",
  ...,
  keep.fraction = 0.1,
  keep.number = Inf,
  keep.sparse = TRUE,
  invert.selection = FALSE,
  h = NULL,
  n = NULL,
  label.fill = "",
  na.rm = TRUE,
  show.legend = FALSE,
  inherit.aes = TRUE
)
```

Arguments

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

invert.selection

> logical If TRUE, the complement of the selected rows are returned.

h

> vector of bandwidths for x and y directions. Defaults to normal reference band-width (see bandwidth.nrd). A scalar value will be taken to apply to both directions.

n

> Number of grid points in each direction. Can be scalar or a length-2 integer vector

label.fill

> character vector of length 1 or a function.

na.rm

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

show.legend

> logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.

inherit.aes

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

Details

stat_dens2d_labels() is designed to work together with statistics from package 'ggrepel'. To avoid text labels being plotted over unlabelled points the corresponding rows in data need to be retained but labels replaced with the empty character string, "". This makes stat_dens2d_filter unsuitable for the task. Non-the-less stat_dens2d_labels() could be useful in some other cases, as the substitution character string can be set by the user.

Value

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

See Also

kde2d used internally.

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(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 fucntion
# instead of a character string.
  label.fun <- function(x) {ifelse(grepl("^a", x), x, "")}
  ggplot(data = d, aes(x, y, label = lab, colour = group)) +
    geom_point() +
    stat_dens2d_labels(geom = "text_repel", label.fill = label.fun)
}
# Using geom_debug() we can see that all 100 rows in \code{d} are
# returned. But only those labelled in the previous example still contain
# the original labels.

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")
}
```

---

stat_fmt_tb | *Select and slice a tibble nested in* data

Description

stat_fmt_tb selects, reorders and/or renames columns and or rows of a tibble nested in data. This stat is intended to be used to pre-process tibble objects mapped to the label aesthetic before adding them to a plot with geom_table.

Usage

```
stat_fmt_tb(
  mapping = NULL,
  data = NULL,
  geom = "table",
  tb.vars = NULL,
  tb.rows = NULL,
  digits = 3,
  position = "identity",
  table.theme = NULL,
  table.rownames = FALSE,
  table.colnames = TRUE,
  table.hjust = 0.5,
  parse = FALSE,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with [aes](#) or [aes_](#). Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
tb.vars, tb.rows	
	character or numeric vectors, optionally named, used to select and/or rename the columns or rows in the table returned.
digits	integer indicating the number of significant digits to be retained in data.
position	The position adjustment to use for overlapping points on this layer
table.theme	NULL, list or function A 'gridExtra' ttheme definition, or a constructor for a ttheme or NULL for default.
table.rownames, table.colnames	
	logical flag to enable or disabling printing of row names and column names.

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

Value

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))

# 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_panel_counts   *Number of observations in a plot panel*

Description

stat_panel_counts() counts the number of observations in each panel. By default it adds a text
label to the top right corner of each panel. Grouping is ignored.

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_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,
  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.

Details

This statistic can be used to automatically count observations in each panel of a plot, and by default add these counts as text labels. This statistic, 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, pass geom = "text" explicitly as argument. A vector with the same length as the number of panels in the figure can be used if needed.

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 per plot quadrant.

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`.

npcx x value of label position in npc units

npcy y value of label position in npc units

count number of observations

x x value of label position in data units

y y value of label position in data units

.

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.

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

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.

See Also

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

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

Examples

```
# generate artificial data
set.seed(67821)
x <- 1:100
y <- rnorm(length(x), mean = 10)
my.data <- data.frame(x, y)

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

# We use geom_debug() 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")
}

ggplot(my.data, aes(x, y)) +
 geom_point() +
 stat_panel_counts(aes(label = sprintf("%i observations", after_stat(count)))) +
 expand_limits(y = 12.7)

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

```
# generate artificial data
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)

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)

# We use geom_debug() 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")

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

}

ggplot(my.data, aes(x, y)) +
 geom_point() +
 stat_panel_counts(aes(label = sprintf("%i observations", after_stat(count)))) +
 expand_limits(y = 12.7)

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

ggplot(my.data, aes(y, colour = group)) +
  stat_group_counts(label.y = "top") +
  stat_density(aes(fill = 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 = "none",
  xintercept = 0,
  yintercept = 0,
  label.x = NULL,
  label.y = NULL,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with [aes](#) or [aes_](#). Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use on this layer
quadrants	integer vector indicating which quadrants are of interest, with a 0L indicating the whole plot.
pool.along	character, one of "none", "x" or "y", indicating which quadrants to pool to calculate counts by pair of quadrants.
xintercept, yintercept	
	numeric the coordinates of the origin of the quadrants.

label.x, label.y

                 numeric Coordinates (in npc units) to be used for absolute positioning of the labels.

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

show.legend   logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.

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

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

Details

This statistic can be used to automatically count observations in each of the four quadrants of a plot, and by default add these counts as text labels. Values exactly equal to xintercept or yintercept are counted together with those larger than the intercepts. An argument value of zero, passed to formal parameter quadrants is interpreted as a request for the count of all observations in each plot panel.

The default origin of quadrants is at xintercept = 0, yintercept = 0. Also by default, counts are computed for all quadrants within the *x* and *y* scale limits, but ignoring any marginal scale expansion. The default positions of the labels is in the farthest corner or edge of each quadrant using npc coordinates. Consequently, when using facets even with free limits for *x* and *y* axes, the location of the labels is consistent across panels. This is achieved by use of geom = "text_npc" or geom = "label_npc". To pass the positions in native data units, pass geom = "text" explicitly as argument.

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

.

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 <- 1:100
y <- rnorm(length(x), mean = 10)
my.data <- data.frame(x, y)

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

# We use geom_debug() to see the computed values

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

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

ggplot(my.data, aes(x, y)) +
 geom_point() +
 stat_quadrant_counts(aes(label = sprintf("%i observations", after_stat(count)))) +
 expand_limits(y = 12.7)

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

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

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

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

```
    stat_quadrant_counts(geom = "text") # use geom_text()
```

---

try_data_frame   *Convert an R object into a tibble*

Description

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

Usage

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

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

Arguments

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

Value

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

Warning!

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

Note

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

Examples

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

---

ttheme_gtdefault   *Table themes*

Description

Additional theme constructors for use with [geom_table](#).

Usage

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

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

ttheme_gtbw(
  base_size = 10,
```

```
    base_colour = "black",
    base_family = "",
    parse = FALSE,
    padding = unit(c(1, 0.6), "char"),
    ...
)

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

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

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

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

ttheme_gtstripes(
    base_size = 10,
    base_colour = "grey10",
    base_family = "",
    parse = FALSE,
```

```
    padding = unit(c(0.8, 0.6), "char"),
    ...
)
```

Arguments

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

Details

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

Value

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

Note

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

Examples

```
library(dplyr)
library(tibble)

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

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

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

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

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

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

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

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

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

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

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

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

---

ttheme_set   *Set default table theme*

Description

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

Usage

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

Arguments

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

Value

A named list with the previous value of the option.

Note

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

Examples

```
library(dplyr)
library(tibble)

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

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

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

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

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

# restore previous setting
ttheme_set(old_ttheme)
```

Index