Package 'ggh4x'

February 11, 2021

Title Hacks for 'ggplot2'

Version 0.1.2.1

Description A 'ggplot2' extension that does a variety of little helpful things. The package extends 'ggplot2' facets through customisation, by setting individual scales per panel, resizing panels and providing nested facets. Also allows multiple colour and fill scales per plot. Also hosts a smaller collection of stats, geoms and axis guides.

License MIT + file LICENSE

URL https://github.com/teunbrand/ggh4x

BugReports https://github.com/teunbrand/ggh4x/issues

Depends ggplot2 (>= 3.3.0)

Imports grid, gtable, scales, vctrs, rlang

Suggests covr, digest, fitdistrplus, ggdendro, knitr, MASS, rmarkdown, stats, testthat (>= 2.1.0), utils

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Collate 'element_part_rect.R' 'facet_nested.R' 'facet_wrap2.R' 'facet_nested_wrap.R' 'facetted_pos_scales.R' 'force_panelsize.R' 'geom_pointpath.R' 'geom_polygonraster.R' 'geom_rectrug.R' 'geom_text_aimed.R' 'ggh4x-package.R' 'ggh4x_extensions.R' 'guide_axis_logticks.R' 'guide_axis_minor.R' 'guide_axis_nested.R' 'guide_axis_truncated.R' 'guide_axis_utils.R' 'guide_dendrogram.R' 'guide_stringlegend.R' 'position_disjoint_ranges.R' 'position_lineartrans.R' 'scale_dendrogram.R' 'scale_listed.R' 'scale_multi.R' 'stat_funxy.R' 'stat_rle.R' 'stat_roll.R' 'stat_theodensity.R' 'themes.R' 'utils.R'

NeedsCompilation no

Author Teun van den Brand [aut, cre] (<https://orcid.org/0000-0002-9335-7468>)

Maintainer Teun van den Brand <tahvdbrand@gmail.com>

Repository CRAN

Date/Publication 2021-02-11 10:00:02 UTC

R topics documented:

center_limits *Center limits*

Description

This a function factory that allows the centering of scales around a certain value while still including all values. Convenient for centering log2 fold change limits around zero.

Usage

```
center_limits(around = 0)
```

Arguments

around	A numeric of length 1 indicating around which value to center the limits.

Value

A function that takes limits and returns expanded limits centered at the around argument.

Examples

```
center_limits(5)(c(3,8))

g <- ggplot(iris,
            aes(Sepal.Width, Sepal.Length,
                colour = log2(Petal.Width / Petal.Length))) +
  geom_point() +
  scale_colour_gradient2(limits = center_limits())
```

element_part_rect *Partial rectangle theme element*

Description

The element_part_rect() function draws sides of a rectangle as theme elements. It can substitute element_rect() theme elements.

Usage

```
element_part_rect(
  side = "tlbr",
  fill = NULL,
  colour = NULL,
  size = NULL,
  linetype = NULL,
  color = NULL,
  inherit.blank = FALSE
)
```

Arguments

side	A character of length one containing any of "t", "l", "b", "r". If these letters are present it will draw an edge at the top (t), left (l), bottom (b) or right (r) respectively. Including all or none of these letters will default to normal element_rect().
fill	Fill colour.
colour	Line/border colour. Color is an alias for colour.
size	Line/border size in mm; text size in pts.
linetype	Line type. An integer (0:8), a name (blank, solid, dashed, dotted, dotdash, longdash, twodash), or a string with an even number (up to eight) of hexadecimal digits which give the lengths in consecutive positions in the string.
color	Line/border colour. Color is an alias for colour.
inherit.blank	Should this element inherit the existence of an element_blank among its parents? If TRUE the existence of a blank element among its parents will cause this element to be blank as well. If FALSE any blank parent element will be ignored when calculating final element state.

Value

An S3 object of class element_part_rect.

Examples

```
ggplot(iris, aes(Sepal.Width, Sepal.Length)) +
  geom_point() +
  facet_grid(Species ~.) +
  theme(
    strip.background = element_part_rect(side = "tb", colour = "black"),
    panel.background = element_part_rect(side = "l", colour = "black")
  )
```

FacetNested *ggh4x extensions to ggplot2*

Description

ggnomics relies on the extension mechanism of ggplot2 through ggproto class objects, which allows cross-package inheritance of objects such as geoms, stats, facets, scales and coordinate systems. These objects can be ignored by users for the purpose of making plots, since interacting with these objects is preferred through various geom_*, stat_*, facet_*, coord_* and scale_* functions.

See Also

ggproto

`facetted_pos_scales` *Set individual scales in facets*

Description

This function allows the tweaking of the position scales (`x` and `y`) of individual facets. You can use it to fine-tune limits, breaks and other scale parameters for individual facets, provided the facet allows free scales.

Usage

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

Arguments

`x, y`	A `list` wherein elements are either x/y position scales or `NULL`s. Alternatively, a list of formulae (see details).

Details

It is intended that this function works with both `facet_wrap` and `facet_grid`. For `facet_wrap`, the scales are used for each individual panel. For `facet_grid`, the scales are used for the rows and columns. Note that these facets must be used with `scales = "free"` or `"free_x"` or `"free_y"`, depending on what scales are added.

Axis titles are derived from the first scale in the list (or the default position scale when the first list element is `NULL`).

Scale transformations: It is allowed to use individual scale transformations for facets, but this functionality comes with the trade-off that the out of bounds (`oob`) argument for individual scales is ignored. Values that are out of bounds will be clipped. Whereas the `stat` part of a ggplot layer is typically calculated after scale transformations, the calculation of the `stat` happens before scale transformation with this function, which can lead to some awkward results. The suggested workaround is to pre-transform the data for layers with non-identity `stat` parts.

Scale list input: `NULL`s are valid list elements and signal that the default position scale should be used at the position in the list where the `NULL` occurs. Since transformations are applied before facet scales are initiated, it is not recommended to use a default position (either the first in the list, or defined outside `facetted_pos_scales()`) scale with a transformation other than `trans = "identity"` (the default).

Formula list input: The `x` and `y` arguments also accept a list of two-sided formulas. The left hand side of a formula should evaluate to a `logical` vector. The right hand side of the formula should evaluate to a position scale, wherein the `x` argument accepts x-position scales and the `y` argument accepts y-position scales. Notably, the left hand side of the formula is evaluated using the tidy evaluation framework, whereby the `data.frame` with the plot's layout is given priority over the environment in which the formula was created. As a consequence, variables (columns) that define faceting groups can be references directly.

Value

A *facetted_pos_scales* object, instructing a ggplot how to adjust the scales per facet.

See Also

scale_continuous and scale_x_discrete.

Examples

```
plot <- ggplot(iris, aes(Sepal.Width, Sepal.Length)) +
  geom_point(aes(colour = Species)) +
  facet_wrap(Species ~ ., scales = "free_y")

# Reversing the y-axis in the second panel. When providing a list of scales,
# NULL indicates to use the default, global scale
plot +
  facetted_pos_scales(
    y = list(NULL, scale_y_continuous(trans = "reverse"))
  )

# Alternative for specifying scales with formula lists. The LHS can access
# columns in the plot's layout.
plot +
  facetted_pos_scales(
    y = list(
      Species == "virginica" ~ scale_y_continuous(breaks = c(6, 7)),
      Species == "versicolor" ~ scale_y_reverse()
    )
  )
```

---

facet_nested *Layout panels in a grid with nested strips*

Description

facet_nested() forms a matrix of panels defined by row and column faceting variables and nests grouped facets.

Usage

```
facet_nested(
  rows = NULL,
  cols = NULL,
  scales = "fixed",
  space = "fixed",
  shrink = TRUE,
  labeller = "label_value",
  as.table = TRUE,
```

```
  switch = NULL,
  drop = TRUE,
  margins = FALSE,
  facets = NULL,
  nest_line = FALSE,
  resect = unit(0, "mm"),
  bleed = FALSE
)
```

Arguments

rows	A set of variables or expressions quoted by `vars()` and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to `labeller`). For compatibility with the classic interface, `rows` can also be a formula with the rows (of the tabular display) on the LHS and the columns (of the tabular display) on the RHS; the dot in the formula is used to indicate there should be no faceting on this dimension (either row or column).
cols	A set of variables or expressions quoted by `vars()` and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to `labeller`). For compatibility with the classic interface, `rows` can also be a formula with the rows (of the tabular display) on the LHS and the columns (of the tabular display) on the RHS; the dot in the formula is used to indicate there should be no faceting on this dimension (either row or column).
scales	Are scales shared across all facets (the default, `"fixed"`), or do they vary across rows (`"free_x"`), columns (`"free_y"`), or both rows and columns (`"free"`)?
space	If `"fixed"`, the default, all panels have the same size. If `"free_y"` their height will be proportional to the length of the y scale; if `"free_x"` their width will be proportional to the length of the x scale; or if `"free"` both height and width will vary. This setting has no effect unless the appropriate scales also vary.
shrink	If `TRUE`, will shrink scales to fit output of statistics, not raw data. If `FALSE`, will be range of raw data before statistical summary.
labeller	A function that takes one data frame of labels and returns a list or data frame of character vectors. Each input column corresponds to one factor. Thus there will be more than one with `vars(cyl,am)`. Each output column gets displayed as one separate line in the strip label. This function should inherit from the "labeller" S3 class for compatibility with `labeller()`. You can use different labeling functions for different kind of labels, for example use `label_parsed()` for formatting facet labels. `label_value()` is used by default, check it for more details and pointers to other options.
as.table	If `TRUE`, the default, the facets are laid out like a table with highest values at the bottom-right. If `FALSE`, the facets are laid out like a plot with the highest value at the top-right.
switch	By default, the labels are displayed on the top and right of the plot. If `"x"`, the top labels will be displayed to the bottom. If `"y"`, the right-hand side labels will be displayed to the left. Can also be set to `"both"`.

drop	If TRUE, the default, all factor levels not used in the data will automatically be dropped. If FALSE, all factor levels will be shown, regardless of whether or not they appear in the data.
margins	Either a logical value or a character vector. Margins are additional facets which contain all the data for each of the possible values of the faceting variables. If FALSE, no additional facets are included (the default). If TRUE, margins are included for all faceting variables. If specified as a character vector, it is the names of variables for which margins are to be created.
facets	This argument is soft-deprecated, please use rows and cols instead.
nest_line	a logical vector of length 1, indicating whether to draw a nesting line to indicate the nesting of variables. Control the look of the nesting line by setting the ggh4x.facet.nestline theme element.
resect	a unit vector of length 1, indicating how much the nesting line should be shortened.
bleed	a logical vector of length 1, indicating whether merging of lower-level variables is allowed when the higher-level variables are separate. See details.

Details

Unlike facet_grid(), this function only automatically expands missing variables when they have no variables in that direction, to allow for unnested variables. It still requires at least one layer to have all faceting variables.

Hierarchies are inferred from the order of variables supplied to rows or cols. The first variable is interpreted to be the outermost variable, while the last variable is interpreted to be the innermost variable. They display order is always such that the outermost variable is placed the furthest away from the panels. Strips are automatically grouped when they span a nested variable.

The bleed argument controls whether lower-level variables are allowed to be merged when higher-level are different, i.e. they can bleed over hierarchies. Suppose the facet_grid() behaviour would be the following:

```
[_1_][_2_][_2_]
[_3_][_3_][_4_]
```

In such case, the default bleed = FALSE argument would result in the following:

```
[_1_][___2____]
[_3_][_3_][_4_]
```

Whereas bleed = TRUE would allow the following:

```
[_1_][___2____]
[___3____][_4_]
```

Value

A *FacetNested* ggproto object.

See Also

See facet_grid for descriptions of the original arguments. See unit for the construction of a unit vector.

Other facetting functions: facet_nested_wrap(), facet_wrap2()

Examples

```
df <- iris
df$nester <- ifelse(df$Species == "setosa",
                    "Short Leaves",
                    "Long Leaves")

ggplot(df, aes(Sepal.Length, Petal.Length)) +
  geom_point() +
  facet_nested(~ nester + Species)

# Controlling the nest line
ggplot(df, aes(Sepal.Length, Petal.Length)) +
  geom_point() +
  facet_nested(~ nester + Species, nest_line = TRUE) +
  theme(ggh4x.facet.nestline = element_line(linetype = 3))
```

---

facet_nested_wrap *Ribbon of panels with nested strips.*

Description

facet_nested_wrap() wraps a sequence of panels onto a two-dimensional layout, and nests grouped facets where possible.

Usage

```
facet_nested_wrap(
  facets,
  nrow = NULL,
  ncol = NULL,
  scales = "fixed",
  axes = "margins",
  remove_labels = "none",
  shrink = TRUE,
  labeller = "label_value",
  as.table = TRUE,
  drop = TRUE,
  dir = "h",
  strip.position = "top",
  nest_line = FALSE,
  resect = unit(0, "mm"),
  bleed = FALSE
)
```

Arguments

facets	A set of variables or expressions quoted by vars() and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to labeller). For compatibility with the classic interface, can also be a formula or character vector. Use either a one sided formula, ~a + b, or a character vector, c("a","b").
nrow	Number of rows and columns.
ncol	Number of rows and columns.
scales	Should scales be fixed ("fixed", the default), free ("free"), or free in one dimension ("free_x", "free_y")?
axes	A character where axes should be drawn. Either "margins" (default), "rows", "cols" or "full". Only applies when the scale is free through the scales argument.
remove_labels	A character denoting what labels should be removed when axes are repeated and redundant. Either "none" (default), "rows", "cols" or "all". Only applies to relevant position guides included with the axes argument when scales are fixed.
shrink	If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.
labeller	A function that takes one data frame of labels and returns a list or data frame of character vectors. Each input column corresponds to one factor. Thus there will be more than one with vars(cyl,am). Each output column gets displayed as one separate line in the strip label. This function should inherit from the "labeller" S3 class for compatibility with labeller(). You can use different labeling functions for different kind of labels, for example use label_parsed() for formatting facet labels. label_value() is used by default, check it for more details and pointers to other options.
as.table	If TRUE, the default, the facets are laid out like a table with highest values at the bottom-right. If FALSE, the facets are laid out like a plot with the highest value at the top-right.
drop	If TRUE, the default, all factor levels not used in the data will automatically be dropped. If FALSE, all factor levels will be shown, regardless of whether or not they appear in the data.
dir	Direction: either "h" for horizontal, the default, or "v", for vertical.
strip.position	By default, the labels are displayed on the top of the plot. Using strip.position it is possible to place the labels on either of the four sides by setting strip.position = c("top","bottom","left","right")
nest_line	a logical vector of length 1, indicating whether to draw a nesting line to indicate the nesting of variables. Control the look of the nesting line by setting the ggh4x.facet.nestline theme element.
resect	a unit vector of length 1, indicating how much the nesting line should be shortened.
bleed	a logical vector of length 1, indicating whether merging of lower-level variables is allowed when the higher-level variables are separate. See details.

Details

This function inherits the capabilities of `facet_wrap2()`.

This function only merges strips in the same row or column as they appear through regular `facet_wrap()` layout behaviour.

Hierarchies are inferred from the order of variables supplied to `rows` or `cols`. The first variable is interpreted to be the outermost variable, while the last variable is interpreted to be the innermost variable. They display order is always such that the outermost variable is placed the furthest away from the panels. Strips are automatically grouped when they span a nested variable.

The `bleed` argument controls whether lower-level variables are allowed to be merged when higher-level are different, i.e. they can bleed over hierarchies. Suppose the `facet_wrap()` behaviour would be the following:

```
[_1_][_2_][_2_]
[_3_][_3_][_4_]
```

In such case, the default `bleed = FALSE` argument would result in the following:

```
[_1_][___2____]
[_3_][_3_][_4_]
```

Whereas `bleed = TRUE` would allow the following:

```
[_1_][___2____]
[___3____][_4_]
```

Value

A `FacetNestedWrap` ggproto object that can be added to a plot.

See Also

Other facetting functions: `facet_nested()`, `facet_wrap2()`

Examples

```
p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point()
p + facet_nested_wrap(vars(cyl, drv))

# Controlling the nest line
p + facet_nested_wrap(vars(cyl, drv), nest_line = TRUE) +
  theme(ggh4x.facet.nestline = element_line(linetype = 3))

# Ignore nested hierarchies with the 'bleed' argument
 p + facet_nested_wrap(vars(drv, cyl), bleed = TRUE)
```

facet_wrap2 *Extended wrapped facets*

Description

This function behaves like `facet_wrap()`, but has a few extra options on axis drawing when scales are fixed.

Usage

```
facet_wrap2(
  facets,
  nrow = NULL,
  ncol = NULL,
  scales = "fixed",
  axes = "margins",
  remove_labels = "none",
  shrink = TRUE,
  labeller = "label_value",
  as.table = TRUE,
  drop = TRUE,
  dir = "h",
  strip.position = "top"
)
```

Arguments

facets	A set of variables or expressions quoted by `vars()` and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to `labeller`). For compatibility with the classic interface, can also be a formula or character vector. Use either a one sided formula, `~a + b`, or a character vector, `c("a","b")`.
nrow	Number of rows and columns.
ncol	Number of rows and columns.
scales	Should scales be fixed (`"fixed"`, the default), free (`"free"`), or free in one dimension (`"free_x"`, `"free_y"`)?
axes	A `character` where axes should be drawn. Either `"margins"` (default), `"rows"`, `"cols"` or `"full"`. Only applies when the scale is free through the `scales` argument.
remove_labels	A `character` denoting what labels should be removed when axes are repeated and redundant. Either `"none"` (default), `"rows"`, `"cols"` or `"all"`. Only applies to relevant position guides included with the `axes` argument when scales are fixed.

shrink	If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.
labeller	A function that takes one data frame of labels and returns a list or data frame of character vectors. Each input column corresponds to one factor. Thus there will be more than one with vars(cyl,am). Each output column gets displayed as one separate line in the strip label. This function should inherit from the "labeller" S3 class for compatibility with labeller(). You can use different labeling functions for different kind of labels, for example use label_parsed() for formatting facet labels. label_value() is used by default, check it for more details and pointers to other options.
as.table	If TRUE, the default, the facets are laid out like a table with highest values at the bottom-right. If FALSE, the facets are laid out like a plot with the highest value at the top-right.
drop	If TRUE, the default, all factor levels not used in the data will automatically be dropped. If FALSE, all factor levels will be shown, regardless of whether or not they appear in the data.
dir	Direction: either "h" for horizontal, the default, or "v", for vertical.
strip.position	By default, the labels are displayed on the top of the plot. Using strip.position it is possible to place the labels on either of the four sides by setting strip.position = c("top","bottom","left","right")

Value

A Facet ggproto object that can be added to a plot.

See Also

Other facetting functions: facet_nested_wrap(), facet_nested()

Examples

```
p <- ggplot(mpg, aes(displ, hwy)) + geom_point()

# Repeat all axes for every facet
p + facet_wrap2(vars(class), axes = "full")

# Repeat only y-axes
p + facet_wrap2(vars(class), axes = "cols")

# Repeat axes without labels
p + facet_wrap2(vars(class), axes = "full", remove_labels = "all")

# Repeat axes without x-axis labels
p + facet_wrap2(vars(class), axes = "full", remove_labels = "rows")
```

`force_panelsizes` *Force a facetted plot to have specified panel sizes*

Description

Takes a ggplot and modifies its facet drawing behaviour such that the widths and heights of panels are set by the user.

Usage

```
force_panelsizes(rows = NULL, cols = NULL, respect = NULL)
```

Arguments

`rows`	a `numeric` or `unit` vector for setting panel heights.
`cols`	a `numeric` or `unit` vector for setting panel widths.
`respect`	a `logical` value. If `TRUE`, widths and heights specified in `"null"` `units` are proportional. If `FALSE`, `"null"` `units` in x- and y-direction vary idependently.

Details

Forcing the panel sizes should in theory work regardless of what facetting choice was made, as long as this function is called after the facet specification. Even when no facets are specified, ggplot2 defaults to the `facet_null` specification; a single panel. `force_panelsizes` works by wrapping the original panel drawing function inside a function that modifies the widths and heights of panel grobs in the original function's output gtable.

When `rows` or `cols` are `numeric` vectors, panel sizes are defined as ratios i.e. relative `"null"` `units`. `rows` and `cols` vectors are repeated or shortened to fit the number of panels in their direction. When `rows` or `cols` are `NULL`, no changes are made in that direction.

When `respect` = `NULL`, default behaviour specified elsewhere is inherited.

No attempt is made to guarantee that the plot fits the output device. The `space` argument in `facet_grid` will be overruled. When individual panels span multiple rows or columns, this function may not work as intended.

Value

A `forcedsize` S3 object that can be added to a plot.

See Also

`facet_grid` `facet_wrap` `facet_null` `unit`

Examples

```
ggplot(mtcars, aes(disp, mpg)) +
  geom_point() +
  facet_grid(vs ~ am) +
  force_panelsizes(rows = c(2, 1),
                   cols = c(2, 1))
```

`geom_pointpath` *Point Paths*

Description

The point path geom is used to make a scatterplot wherein the points are connected with lines in some order. This geom intends to mimick the `type = 'b'` style of base R line plots.

Usage

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

Arguments

`mapping`	Set of aesthetic mappings created by `aes()` or `aes_()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created. A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x,10)`).
`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(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
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. It can also be a named logical vector to finely select the aesthetics to display.
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 linesize aesthetic can be interpreted as the size aesthetic for geom_line(). The mult is a numeric value to scale the proportion of gaps in the line around points.

While the need for this geom is not very aparrent, since it can be approximated in a variety of ways, the trick up its sleeve is that it dynamically adapts the interpoint segments so these don't deform under different aspect ratios or device sizes.

Value

A *Layer* ggproto object.

Aesthetics

geom_pointpath() understands the following aesthetics (required aesthetics are in bold):

- x
- y
- alpha
- colour
- group
- shape
- size
- stroke
- linesize
- linetype
- mult

Examples

```
ggplot(pressure, aes(temperature, pressure)) +
  geom_pointpath()
```

geom_polygonraster *Polygon parameterisation for rasters*

Description

geom_polygonraster takes data that describes a raster with pixels of the same size and reparametrises the data as a polygon. This allows for more flexible transformations of the data, but comes at an efficiency cost.

Usage

```
geom_polygonraster(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = position_lineartrans(),
  ...,
  hjust = 0.5,
  vjust = 0.5,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)).
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(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.

hjust	horizontal and vertical justification of the grob. Each justification value should be a number between 0 and 1. Defaults to 0.5 for both, centering each pixel over its data location.
vjust	horizontal and vertical justification of the grob. Each justification value should be a number between 0 and 1. Defaults to 0.5 for both, centering each pixel over its data location.
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
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. It can also be a named logical vector to finely select the aesthetics to display.
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

For each pixel in a raster, makes a vertex for each of the four corner points. These coordinates can then by transformed by coord-functions such as coord_polar or position-functions such as position_lineartrans. Currently substitutes group aesthetics right before drawing in favour of pixel identifiers.

Value

A *Layer* ggproto object.

Aesthetics

geom_raster() understands the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- fill
- alpha
- group

See Also

geom_raster

Examples

```
# Combining with coord_polar()
ggplot(faithfuld, aes(waiting, eruptions)) +
  geom_polygonraster(aes(fill = density)) +
  coord_polar()

# Combining with linear transformations
```

```
df <- data.frame(x = row(volcano)[TRUE],
                 y = col(volcano)[TRUE],
                 z = volcano[TRUE])

ggplot(df, aes(x, y, fill = z)) +
  geom_polygonraster(position = position_lineartrans(angle = 30,
                                                     shear = c(1, 0)))
```

`geom_rectmargin` *Rectangular rugs in the margins*

Description

Like rug plots display data points of a 2D plot as lines in the margins, this function plots rectangles in the margins. Rectangular rugs are convenient for displaying onedimensional, ranged annotations for twodimensional plots.

Usage

```
geom_rectmargin(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  outside = FALSE,
  sides = "bl",
  length = unit(0.03, "npc"),
  linejoin = "mitre",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_tilemargin(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  outside = FALSE,
  sides = "bl",
  length = unit(0.03, "npc"),
  linejoin = "mitre",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)).
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(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.
outside	logical of length 1 that controls wether to move the rectangles outside of the plot area. For the best results, it is probably best to set coord_cartesian(clip = "off") and avoid overlap with the default axes by changing the sides argument to "tr".
sides	A string of length 1 that controls which sides of the plot the rug-rectangles appear on. A string containing any letters in "trbl" will set it to top, right, bottom and left respectively.
length	A unit object that sets the width and height of the rectangles in the x- and y-directions respectively. Note that scale expansion can affect the look of this.
linejoin	Line join style (round, mitre, bevel).
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
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. It can also be a named logical vector to finely select the aesthetics to display.
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

By default, scales are expanded 5% on either side of the plot, whereas the rug rectangles will occupy 3% of the total plot size by default. The geom_rectmargin() and geom_tilemargin() versions do the same thing, but are parameterised differently; see geom_rect.

These functions do not have hardcoded required aesthetics, since the x and y directions can be omitted by not choosing a side in the corresponding direction, i.e. y-direction variables are omitted

when plotting the rug only on the top and/or bottom. This can result in errors when the aesthetics are not specified appropriately, so some caution is advised.

Value

A *Layer* ggproto object.

Aesthetics

`geom_rectmargin()` requires either one of the following sets of aesthetics, but also can use both:

- **xmin**
- **xmax**

and/or:

- **ymin**
- **ymax**

`geom_tilemargin()` requires either one of the following sets of aesthetics, but can also use both:

- **x**
- **width**

and/or:

- **y**
- **height**

Furthermore, `geom_rectmargin()` and `geom_tilemargin()` also understand these shared aesthetics:

- alpha
- colour
- fill
- group
- linetype
- size

See Also

`geom_rug`, `geom_rect`, `geom_tile`

Examples

```
# geom_rectmargin() is parameterised by the four corners
df <- data.frame(
  xmin = c(1, 5),
  xmax = c(2, 7),
  ymin = c(1, 2),
  ymax = c(2, 4),
  fill = c("A", "B")
)


ggplot(df, aes(xmin = xmin, xmax = xmax,
               ymin = ymin, ymax = ymax,
               fill = fill)) +
  geom_rect() +
  geom_rectmargin()

# geom_tilemargin() is parameterised by center and size
df <- data.frame(
  x = c(1, 4),
  y = c(1, 2),
  width = c(2, 1),
  height = c(1, 2),
  fill = c("A", "B")
)

ggplot(df, aes(x, y,
               width = width, height = height,
               fill = fill)) +
  geom_tile() +
  geom_tilemargin()
```

---

geom_text_aimed *Aimed text*

Description

Similar to `geom_text()`, this geom also generates text but places the text at an angle so that the text seems aimed towards a point defined by `[xend,yend]`.

Usage

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

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

Arguments

mapping	Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)).
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. Cannot be jointly specified with nudge_x or nudge_y.
...	Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.
parse	If TRUE, the labels will be parsed into expressions and displayed as described in ?plotmath.
nudge_x	Horizontal and vertical adjustment to nudge labels by. Useful for offsetting text from points, particularly on discrete scales. Cannot be jointly specified with position.
nudge_y	Horizontal and vertical adjustment to nudge labels by. Useful for offsetting text from points, particularly on discrete scales. Cannot be jointly specified with position.
flip_upsidedown	A logical(1). If TRUE (default), the angle of text placed at angles between 90 and 270 degrees is flipped so that it is more comfortable to read. If FALSE, will take calculated angles literally.
check_overlap	If TRUE, text that overlaps previous text in the same layer will not be plotted. check_overlap happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling geom_label() or geom_text().

na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
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. It can also be a named logical vector to finely select the aesthetics to display.
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 calculated angle is such that the text will be parallel to a line passing through the coordinates [x,y] and [xend,yend]. The calculated angle is added to the angle angle aesthetic, so that you can set text perpendicular to that line by setting angle = 90. These angles are calculated in absolute coordinates, meaning that resizing the plot will retain the same appearance.

Value

A ggplot2 Layer

Aesthetics

geom_text_aimed() understands the following aesthetics (required aesthetics are in bold):

- x
- y
- label
- alpha
- angle
- colour
- family
- fontface
- group
- hjust
- lineheight
- size
- vjust
- xend
- yend

Learn more about setting these aesthetics in vignette("ggplot2-specs").

Note

When using this geom to aim text at the centre of a polar plot, make sure the radius range does not have close to zero width.

Examples

```
# Point all labels to upper right corner
ggplot(mtcars, aes(mpg, wt)) +
  geom_text_aimed(aes(label = rownames(mtcars)),
                  xend = Inf, yend = Inf)

# Point all labels to center of polar plot
ggplot(mpg, aes(manufacturer)) +
  geom_bar(width = 1, aes(fill = manufacturer), show.legend = FALSE) +
  geom_text_aimed(aes(label = manufacturer), hjust = 0,
                  stat = "count", nudge_y = 2) +
  scale_x_discrete(labels = NULL) +
  coord_polar()
```

---

ggsubset *Passing a subset of data to ggplot2 layers.*

Description

This is a convenience function to allow layer objects, such as geoms, to take a subset of the data in the main `ggplot()` call, without storing a duplicate of the subset in the ggplot object.

Usage

```
ggsubset(rowtest = NULL, omit = NULL)
```

Arguments

`rowtest`	logical `expression` indicating which rows to keep.
`omit`	a `character` column name to exclude.

Details

`ggsubset` is a wrapper around `subset.data.frame` where the `subset` argument is set to `rowtest` and the `select` argument to `-omit`. Since the `data` argument in the `layer()` function can take a function with one argument, we can pass the function returned from `ggsubset` as that argument to subset the data by rows.

Value

A function that takes a `data.frame` as argument and returns a subset of that `data.frame` according to `rowtest`

See Also

See `layer`, specifically the `data` argument. See `subset.data.frame` for the internal function.

Examples

```
ggplot(iris, aes(Sepal.Width, Sepal.Length)) +
  geom_point(data = ggsubset(Species == "setosa"))
```

`guide_axis_logticks` *Axis guide with ticks for logarithmic breaks*

Description

This axis guide is probably best described as `annotation_logticks` but implemented as a guide instead of a geom. The tick marks probably best suit log10 transformations.

Usage

```
guide_axis_logticks(
  title = waiver(),
  check.overlap = FALSE,
  angle = NULL,
  n.dodge = 1,
  order = 0,
  position = waiver(),
  prescaled = FALSE,
  trunc_lower = NULL,
  trunc_upper = NULL,
  base = waiver()
)
```

Arguments

`title`	A character string or expression indicating a title of guide. If `NULL`, the title is not shown. By default (`waiver()`), the name of the scale object or the name specified in `labs()` is used for the title.
`check.overlap`	silently remove overlapping labels, (recursively) prioritizing the first, last, and middle labels.
`angle`	Compared to setting the angle in `theme()` / `element_text()`, this also uses some heuristics to automatically pick the `hjust` and `vjust` that you probably want.
`n.dodge`	The number of rows (for vertical axes) or columns (for horizontal axes) that should be used to render the labels. This is useful for displaying labels that would otherwise overlap.
`order`	Used to determine the order of the guides (left-to-right, top-to-bottom), if more than one guide must be drawn at the same location.
`position`	Where this guide should be drawn: one of top, bottom, left, or right.
`prescaled`	A `logical` of length one, indicating wether the data has been manually rescaled (`TRUE`) or the scale takes care of the transformation (`FALSE`).

`trunc_lower` The lower and upper range of the truncated axis:

- `NULL` to not perform any truncation.
- A `function` that takes the break positions as input and returns the lower or upper boundary. Note that also for discrete scales, positions are the mapped positions as `numeric`.
- A `numeric` value in data units for the lower and upper boundaries.
- A `unit` object.

`trunc_upper` The lower and upper range of the truncated axis:

- `NULL` to not perform any truncation.
- A `function` that takes the break positions as input and returns the lower or upper boundary. Note that also for discrete scales, positions are the mapped positions as `numeric`.
- A `numeric` value in data units for the lower and upper boundaries.
- A `unit` object.

`base` When this is provided, the guide takes this as the base for the log transformation instead of trying to guess the base. It is recommended to use this argument if the base is not `10`.

Details

The length of minor ticks can be controlled relative to the length of major ticks by setting `ggh4x.axis.ticks.length.minor` as a `rel` object. Likewise the size of the smallest ticks are controlled by the `ggh4x.axis.ticks.length.minor`, also relative to the major ticks.

Value

An *axis_logticks* guide class object.

See Also

Other axis-guides: `guide_axis_minor()`, `guide_axis_nested()`, `guide_axis_truncated()`

Examples

```
# The guide works well out of the box with log10 scales
p <- ggplot(pressure, aes(temperature, pressure)) +
  geom_line()
p + scale_y_log10(guide = "axis_logticks")

# If the data is already transformed, you can set 'prescaled' to TRUE
ggplot(pressure, aes(temperature, log10(pressure))) +
  geom_line() +
  guides(y = guide_axis_logticks(prescaled = TRUE))

# The lenghts of the log-ticks are controlled by the theme relative to the
# major ticks.
p + scale_y_log10(guide = "axis_logticks") +
  theme(
```

```
    axis.ticks.length.y = unit(1, "cm"),
    ggh4x.axis.ticks.length.minor = rel(0.55),
    ggh4x.axis.ticks.length.mini = rel(0.1)
  )
```

`guide_axis_minor` *Axis guide with ticks for minor breaks*

Description

These are similar the the normal axis guides for position scales, but also place tickmarks at minor break positions.

Usage

```
guide_axis_minor(
  title = waiver(),
  check.overlap = FALSE,
  angle = NULL,
  n.dodge = 1,
  order = 0,
  trunc_lower = NULL,
  trunc_upper = NULL,
  position = waiver()
)
```

Arguments

`title`	A character string or expression indicating a title of guide. If `NULL`, the title is not shown. By default (`waiver()`), the name of the scale object or the name specified in `labs()` is used for the title.
`check.overlap`	silently remove overlapping labels, (recursively) prioritizing the first, last, and middle labels.
`angle`	Compared to setting the angle in `theme()` / `element_text()`, this also uses some heuristics to automatically pick the `hjust` and `vjust` that you probably want.
`n.dodge`	The number of rows (for vertical axes) or columns (for horizontal axes) that should be used to render the labels. This is useful for displaying labels that would otherwise overlap.
`order`	Used to determine the order of the guides (left-to-right, top-to-bottom), if more than one guide must be drawn at the same location.
`trunc_lower`	The lower and upper range of the truncated axis: • `NULL` to not perform any truncation. • A `function` that takes the break positions as input and returns the lower or upper boundary. Note that also for discrete scales, positions are the mapped positions as `numeric`.

- A `numeric` value in data units for the lower and upper boundaries.
- A `unit` object.

`trunc_upper` The lower and upper range of the truncated axis:

- `NULL` to not perform any truncation.
- A `function` that takes the break positions as input and returns the lower or upper boundary. Note that also for discrete scales, positions are the mapped positions as `numeric`.
- A `numeric` value in data units for the lower and upper boundaries.
- A `unit` object.

`position` Where this guide should be drawn: one of top, bottom, left, or right.

Details

The length of minor ticks can be controlled relative to the length of major ticks by setting `ggh4x.axis.ticks.length.minor` as a `rel` object.

Value

An *axis_minor* guide class object.

See Also

Other axis-guides: `guide_axis_logticks()`, `guide_axis_nested()`, `guide_axis_truncated()`

Examples

```
# Using the minor breaks axis
p <- ggplot(iris, aes(Sepal.Width, Sepal.Length)) +
  geom_point()
p + scale_y_continuous(guide = "axis_minor")

# Minor break positions are still controlled by the scale
p + scale_y_continuous(guide = "axis_minor",
                       minor_breaks = seq(4, 8, by = 0.2))

# Minor tick length is controlled relative to major ticks
p + scale_y_continuous(guide = "axis_minor") +
  theme(ggh4x.axis.ticks.length.minor = rel(0.1))
```

---

guide_axis_nested *Nested axis guide*

Description

Discrete position scales containing interacting factors can be visualised more clearly with a nested axis guide. Nested axis guides seperate labels based on a delimiter and groups identical later labels, indicating the grouping with a line spanning the earlier labels.

Usage

```
guide_axis_nested(
  title = waiver(),
  check.overlap = FALSE,
  angle = NULL,
  n.dodge = 1,
  order = 0,
  position = waiver(),
  delim = waiver(),
  trunc_lower = NULL,
  trunc_upper = NULL,
  extend = 0.5
)
```

Arguments

title	A character string or expression indicating a title of guide. If NULL, the title is not shown. By default (waiver()), the name of the scale object or the name specified in labs() is used for the title.
check.overlap	silently remove overlapping labels, (recursively) prioritizing the first, last, and middle labels.
angle	Compared to setting the angle in theme() / element_text(), this also uses some heuristics to automatically pick the hjust and vjust that you probably want.
n.dodge	The number of rows (for vertical axes) or columns (for horizontal axes) that should be used to render the labels. This is useful for displaying labels that would otherwise overlap.
order	Used to determine the order of the guides (left-to-right, top-to-bottom), if more than one guide must be drawn at the same location.
position	Where this guide should be drawn: one of top, bottom, left, or right.
delim	A character of length 1 to tell strsplit how hierarchies should be broken up. Internally defaults to "." to match interaction's default delimiter.
trunc_lower	The lower and upper range of the truncated axis: • NULL to not perform any truncation. • A function that takes the break positions as input and returns the lower or upper boundary. Note that also for discrete scales, positions are the mapped positions as numeric. • A numeric value in data units for the lower and upper boundaries. • A unit object.
trunc_upper	The lower and upper range of the truncated axis: • NULL to not perform any truncation. • A function that takes the break positions as input and returns the lower or upper boundary. Note that also for discrete scales, positions are the mapped positions as numeric.

- A numeric value in data units for the lower and upper boundaries.
- A unit object.

extend A numeric of length 1 indicating how much to extend nesting lines relative to the smallest difference in break positions.

Details

The guide itself makes no effort to group and order labels. To get nice groupings, consider re-ordering the levels of factor variables, or try setting the 'breaks' argument of a scale appropriately.

Value

A *axis_nested* guide class object.

See Also

guide_axis for the classic axis documentation.
weave_factors for an alternative to interaction().

Other axis-guides: guide_axis_logticks(), guide_axis_minor(), guide_axis_truncated()

Examples

```
# The defaults are suited for interaction variables
ggplot(mpg, aes(interaction(cyl, class), hwy)) +
  geom_boxplot() +
  scale_x_discrete(guide = "axis_nested")

# Control where labels are cut with the delim argument
ggplot(mpg, aes(interaction(cyl, class, sep = "~!~"), hwy)) +
  geom_boxplot() +
  scale_x_discrete(guide = guide_axis_nested(delim = "!"))

# The nesting lines inherit looks from axis ticks
ggplot(mpg, aes(interaction(cyl, class), hwy)) +
  geom_boxplot() +
  scale_x_discrete(guide = "axis_nested") +
  theme(axis.ticks = element_line(colour = "red"))

# The looks can be controlled independently by setting `ggh4x.axis.nestline`
ggplot(mpg, aes(interaction(cyl, class), hwy)) +
  geom_boxplot() +
  scale_x_discrete(guide = "axis_nested") +
  theme(ggh4x.axis.nestline = element_line(linetype = 2))
```

guide_axis_truncated *Axis guide with truncated line.*

Description

This axis guide is similar to the normal axis guides for position scales, but can shorten the axis line that is being drawn.

Usage

```
guide_axis_truncated(
  title = waiver(),
  check.overlap = FALSE,
  angle = NULL,
  n.dodge = 1,
  order = 0,
  trunc_lower = min,
  trunc_upper = max,
  position = waiver()
)
```

Arguments

title	A character string or expression indicating a title of guide. If NULL, the title is not shown. By default (waiver()), the name of the scale object or the name specified in labs() is used for the title.
check.overlap	silently remove overlapping labels, (recursively) prioritizing the first, last, and middle labels.
angle	Compared to setting the angle in theme() / element_text(), this also uses some heuristics to automatically pick the hjust and vjust that you probably want.
n.dodge	The number of rows (for vertical axes) or columns (for horizontal axes) that should be used to render the labels. This is useful for displaying labels that would otherwise overlap.
order	Used to determine the order of the guides (left-to-right, top-to-bottom), if more than one guide must be drawn at the same location.
trunc_lower, trunc_upper	The lower and upper range of the truncated axis: • NULL to not perform any truncation. • A function that takes the break positions as input and returns the lower or upper boundary. Note that also for discrete scales, positions are the mapped positions as numeric. • A numeric value in data units for the lower and upper boundaries. • A unit object.
position	Where this guide should be drawn: one of top, bottom, left, or right.

Value

An *axis_truncated* guide class object.

See Also

Other axis-guides: guide_axis_logticks(), guide_axis_minor(), guide_axis_nested()

Examples

```
# Make a plot
p <- ggplot(mtcars, aes(wt, mpg)) +
  geom_point() +
  theme(axis.line = element_line(colour = "black"))

# Setting the default truncated axis
p + guides(x = "axis_truncated")

# Truncating in data units
p + guides(x = guide_axis_truncated(
  trunc_lower = 2.5, trunc_upper = 4.5
))

# Truncate by setting units
p + guides(x = guide_axis_truncated(
  trunc_lower = unit(0.1, "npc"),
  trunc_upper = unit(0.9, "npc")
))

# Truncating with functions
p + guides(x = guide_axis_truncated(
  trunc_lower = function(x) {x - 0.2},
  trunc_upper = function(x) {x + 0.2}
))
```

---

guide_dendro *Dendrogram guide*

Description

Visual representation of a discrete variable with hierarchical relationships between members, like those detailed in scale_(x|y)_dendrogram).

Usage

```
guide_dendro(
  title = waiver(),
  check.overlap = FALSE,
  n.dodge = 1,
  order = 0,
```

```
  position = waiver(),
  label = TRUE,
  trunc_lower = NULL,
  trunc_upper = NULL,
  dendro = waiver()
)
```

Arguments

`title`	A character string or expression indicating a title of guide. If `NULL`, the title is not shown. By default (`waiver()`), the name of the scale object or the name specified in `labs()` is used for the title.
`check.overlap`	silently remove overlapping labels, (recursively) prioritizing the first, last, and middle labels.
`n.dodge`	The number of rows (for vertical axes) or columns (for horizontal axes) that should be used to render the labels. This is useful for displaying labels that would otherwise overlap.
`order`	Used to determine the order of the guides (left-to-right, top-to-bottom), if more than one guide must be drawn at the same location.
`position`	Where this guide should be drawn: one of top, bottom, left, or right.
`label`	A `logical(1)`. If `TRUE`, labels are drawn at the dendrogram leaves. If `FALSE`, labels are not drawn.
`trunc_lower`	The lower and upper range of the truncated axis: • `NULL` to not perform any truncation. • A `function` that takes the break positions as input and returns the lower or upper boundary. Note that also for discrete scales, positions are the mapped positions as `numeric`. • A `numeric` value in data units for the lower and upper boundaries. • A `unit` object.
`trunc_upper`	The lower and upper range of the truncated axis: • `NULL` to not perform any truncation. • A `function` that takes the break positions as input and returns the lower or upper boundary. Note that also for discrete scales, positions are the mapped positions as `numeric`. • A `numeric` value in data units for the lower and upper boundaries. • A `unit` object.
`dendro`	Relevant plotting data for a dendrogram such as those returned by `dendro_data`.

Details

The dendrogram guide inherits graphical elements from the `axis.ticks` theme element. However, the size of the dendrogram is set to 10 times the `axis.ticks.length` theme element.

Value

A *dendroguide* class object.

Examples

```
clust <- hclust(dist(USArrests), "ave")

# Melting USArrests
df <- data.frame(
  State = rownames(USArrests)[row(USArrests)],
  variable = colnames(USArrests)[col(USArrests)],
  value = unname(do.call(c, USArrests))
)

# The guide function can be used to customise the axis
g <- ggplot(df, aes(variable, State, fill = value)) +
  geom_raster() +
  scale_y_dendrogram(hclust = clust,
                     guide = guide_dendro(n.dodge = 2))

# The looks of the dendrogram are controlled through ticks
g + theme(axis.ticks = element_line(colour = "red"))

# The size of the dendrogram is controlled through tick size * 10
g + theme(axis.ticks.length = unit(5, "pt"))
```

---

guide_stringlegend *String legend*

Description

This type of legend shows colour and fill mappings as coloured text. It does not draw keys as guide_legend() does.

Usage

```
guide_stringlegend(
  title = waiver(),
  title.position = NULL,
  title.theme = NULL,
  title.hjust = NULL,
  title.vjust = NULL,
  label.theme = NULL,
  label.hjust = NULL,
  label.vjust = NULL,
  family = NULL,
  face = NULL,
  size = NULL,
  spacing.x = NULL,
  spacing.y = NULL,
  spacing = NULL,
  default.units = "pt",
```

```
  direction = NULL,
  nrow = NULL,
  ncol = NULL,
  byrow = FALSE,
  reverse = FALSE,
  order = 0,
  ...
)
```

Arguments

title	A character string or expression indicating a title of guide. If NULL, the title is not shown. By default (waiver()), the name of the scale object or the name specified in labs() is used for the title.
title.position	A character string indicating the position of a title. One of "top" (default for a vertical guide), "bottom", "left" (default for a horizontal guide), or "right."
title.theme	A theme object for rendering the title text. Usually the object of element_text() is expected. By default, the theme is specified by legend.title in theme() or theme.
title.hjust	A number specifying horizontal justification of the title text.
title.vjust	A number specifying vertical justification of the title text.
label.theme	A theme object for rendering the label text. Usually the object of element_text() is expected. By default, the theme is specified by legend.text in theme().
label.hjust	A numeric specifying horizontal justification of the label text.
label.vjust	A numeric specifying vertical justification of the label text.
family	A character(1) setting a font family for labels.
face	A character(1) setting a font face for labels. One of the following: "plain", "italic" or "bold", "bold.italic".
size	A numeric(1) setting the label text size in pts.
spacing.x, spacing.y, spacing	A numeric(1) or unit for the spacing between label rows and columns. Internally defaults to half the size of the title.
default.units	A character(1) indicating the default units to use if the spacing.* arguments are only given as numeric vectors.
direction	A character string indicating the direction of the guide. One of "horizontal" or "vertical."
nrow	The desired number of rows of legends.
ncol	The desired number of column of legends.
byrow	logical. If FALSE (the default) the legend-matrix is filled by columns, otherwise the legend-matrix is filled by rows.
reverse	logical. If TRUE the order of legends is reversed.
order	positive integer less than 99 that specifies the order of this guide among multiple guides. This controls the order in which multiple guides are displayed, not the contents of the guide itself. If 0 (default), the order is determined by a secret algorithm.
...	ignored.

Value

A guide, stringlegend S3 object.

Examples

```
p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = manufacturer))

# String legend can be set in the `guides()` function
p + guides(colour = guide_stringlegend(ncol = 2))

# The string legend can also be set as argument to the scale
p + scale_colour_viridis_d(guide = "stringlegend")
```

---

position_disjoint_ranges

Segregrating overlapping ranges

Description

One-dimensional ranged data in the x-direction is segregated in the y-direction such that no overlap in twodimensional space occurs. This positioning works best when no relevant information is plotted in the y-direction.

Usage

```
position_disjoint_ranges(extend = 1, stepsize = 1)
```

Arguments

extend	a numeric of length 1 indicating how far a range should be extended in total for calculating overlaps. Setting this argument to a positive number leaves some space between ranges in the same bin.
stepsize	a numeric of length 1 that determines how much space is added between bins in the y-direction. A positive value grows the bins from bottom to top, while a negative value grows the bins from top to bottom.

Details

An object is considered disjoint from a second object when the range between their xmin and xmax coordinates don't overlap. Objects that overlap are assigned to different bins in the y-direction, whereby lower bins are filled first. This way, information in the x-direction is preserved and different objects can be discerned.

Note that this positioning is only particularly useful when y-coordinates do not encode relevant information. Geoms that pair well with this positioning are geom_rect and geom_tile.

This positioning function was inspired by the disjointBins() function in the IRanges package, but has been written such that it accepts any numeric input next to solely integer input.

Value

A *PositionDisjointRanges* object.

See Also

The `disjointBins` function the Bioconductor IRanges package.

Examples

```
# Even though geom_tile() is parametrised by middle-x values, it is
# internally converted to xmin, xmax, ymin, ymax parametrisation so the
# positioning still works.

ggplot() +
  geom_tile(aes(x = rnorm(200), y = 0),
            width = 0.2, height = 0.9,
            position = position_disjoint_ranges(extend = 0.1))
```

`position_lineartrans` *Linearly transform coordinates*

Description

Transforms coordinates in two dimensions in a linear manner for layers that have an `x` and `y` parametrisation.

Usage

```
position_lineartrans(scale = c(1, 1), shear = c(0, 0), angle = 0, M = NULL)
```

Arguments

`scale`	A `numeric` of length two describing relative units with which to multiply the `x` and `y` coordinates respectively.
`shear`	A `numeric` of length two giving relavite units by which to shear the output. The first number is for vertical shearing whereas the second is for horizontal shearing.
`angle`	A `numeric` noting an angle in degrees by which to rotate the input clockwise.
`M`	A `2 x 2` real `matrix`: the transformation matrix for linear mapping. Overrides other arguments if provided.

Details

Linear transformation matrices are `2 x 2` real matrices. The '`scale`', '`shear`' and '`rotation`' arguments are convenience arguments to construct a transformation matrix. These operations occur in the order: scaling - shearing - rotating. To apply the transformations in another order, build a custom '`M`' argument.

For some common transformations, you can find appropriate matrices for the '`M`' argument below.

Value

A *PositionLinearTrans* ggproto object.

Common transformations

Identity transformations: An identity transformation, or returning the original coordinates, can be performed by using the following transformation matrix:

```
| 1 0 |
| 0 1 |
```

or

```
M <-matrix(c(1,0,0,1),2)
```

Scaling: A scaling transformation multiplies the dimension of an object by some amount. An example transformation matrix for scaling everything by a factor 2:

```
| 2 0 |
| 0 2 |
```

or

```
M <-matrix(c(2,0,0,2),2)
```

Squeezing: Similar to scaling, squeezing multiplies the dimensions by some amount that is unequal for the x and y coordinates. For example, squeezing y by half and expanding x by two:

```
|   2   0     |
|   0   0.5   |
```

or

```
M <-matrix(c(2,0,0,0.5),2)
```

Reflection: Mirroring the coordinates around one of the axes. Reflecting around the x-axis:

```
|   1    0   |
|   0   -1   |
```

or

```
M <-matrix(c(1,0,0,-1),2)
```

Reflecting around the y-axis:

```
|   -1   0   |
```

```
|     0     1     |
```

or

```
M <-matrix(c(-1,0,0,1),2)
```

Projection: For projecting the coordinates on one of the axes, while collapsing everything from the other axis. Projecting onto the y-axis:

```
|     0     0     |
|     0     1     |
```

or

```
M <-matrix(c(0,0,0,1),2)
```

Projecting onto the x-axis:

```
|     1     0     |
|     0     0     |
```

or

```
M <-matrix(c(1,0,0,0),2)
```

Shearing: Tilting the coordinates horizontally or vertically. Shearing vertically by 10%:

```
|     1       0     |
|     0.1     1     |
```

or

```
M <-matrix(c(1,0.1,0,1),2)
```

Shearing horizontally by 200%:

```
|     1     2     |
|     0     1     |
```

or

```
M <-matrix(c(1,0,2,1),2)
```

Rotation: A rotation performs a motion around a fixed point, typically the origin the coordinate

system. To rotate the coordinates by 90 degrees counterclockwise:

```
|     0     -1     |
|     1      0     |
```

or

```
M <-matrix(c(0,1,-1,0),2)
```

For a rotation around any angle θ :

$$\begin{vmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{vmatrix}$$

or

```
M <-matrix(c(cos(theta),sin(theta),-sin(theta),cos(theta)),2)
```
with 'theta' defined in radians.

Examples

```
df <- data.frame(x = c(0, 1, 1, 0),
                 y = c(0, 0, 1, 1))
ggplot(df, aes(x, y)) +
  geom_polygon(position = position_lineartrans(angle = 30))

# Custom transformation matrices
# Rotation
theta <- -30 * pi / 180
rot <- matrix(c(cos(theta), sin(theta), -sin(theta), cos(theta)), 2)
# Shear
shear <- matrix(c(1, 0, 1, 1), 2)

# Shear and then rotate
M <- rot %*% shear
ggplot(df, aes(x, y)) +
  geom_polygon(position = position_lineartrans(M = M))
# Alternative shear and then rotate
ggplot(df, aes(x, y)) +
  geom_polygon(position = position_lineartrans(shear = c(0, 1), angle = 30))

# Rotate and then shear
M <- shear %*% rot
ggplot(df, aes(x, y)) +
  geom_polygon(position = position_lineartrans(M = M))
```

scale_dendrogram *Dendrogram position scales*

Description

When discrete data has some inherent hierarchy to the relationship between discrete categories, you can display a dendrogram instead of a tick axis.

Usage

```
scale_x_dendrogram(
  ...,
  hclust = waiver(),
  expand = waiver(),
  guide = waiver(),
  position = "bottom"
)

scale_y_dendrogram(
  ...,
  hclust = waiver(),
  expand = waiver(),
  guide = waiver(),
  position = "left"
)
```

Arguments

`...` Arguments passed on to `ggplot2::discrete_scale`

`aesthetics` The names of the aesthetics that this scale works with.

`scale_name` The name of the scale that should be used for error messages associated with this scale.

`palette` A palette function that when called with a single integer argument (the number of levels in the scale) returns the values that they should take (e.g., `scales::hue_pal()`).

`name` The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

`labels` One of:

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- A function that takes the breaks as input and returns labels as output

`limits` One of:

- `NULL` to use the default scale values

- A character vector that defines possible values of the scale and their order
- A function that accepts the existing (automatic) values and returns new ones

`na.translate` Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify `na.translate = FALSE`.

`na.value` If `na.translate = TRUE`, what aesthetic value should the missing values be displayed as? Does not apply to position scales where `NA` is always placed at the far right.

`drop` Should unused factor levels be omitted from the scale? The default, `TRUE`, uses the levels that appear in the data; `FALSE` uses all the levels in the factor.

`super` The super class to use for the constructed scale

`hclust` An object of the type produced by the `hclust` function.

`expand` For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function `expansion()` to generate the values for the expand argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

`guide` A function used to create a guide or its name. See `guides()` for more information.

`position` For position scales, The position of the axis. `left` or `right` for y axes, `top` or `bottom` for x axes.

Details

The dendrogram type of scale does two things, first it reorders the values along the relevant direction such that they follow the order captured in the `hclust` argument. Secondly, it draws the dendrogram at the axis. The dendrogram visuals inherit from the ticks theme elements, so defining a linetype for the tick marks sets the linetype for the dendrogram.

Value

A *ScaleDendrogram* ggproto object.

Examples

```
# Hierarchically cluster USArrests
yclus <- hclust(dist(USArrests), "ave")
xclus <- hclust(dist(t(USArrests)), "ave")

# Melting USArrests
df <- data.frame(
  State = rownames(USArrests)[row(USArrests)],
  variable = colnames(USArrests)[col(USArrests)],
  value = unname(do.call(c, USArrests))
)
```

```
# Supply the clustering to the scales
ggplot(df, aes(variable, State, fill = value)) +
  geom_raster() +
  scale_y_dendrogram(hclust = yclus) +
  scale_x_dendrogram(hclust = xclus)
```

`scale_fill_multi` *Multiple gradient colour scales*

Description

Maps multiple aesthetics to multiple colour fill gradient scales. It takes in listed arguments for each aesthetic and disseminates these to `continuous_scale`.

Usage

```
scale_fill_multi(
  ...,
  colours,
  values = NULL,
  na.value = "transparent",
  guide = "colourbar",
  aesthetics = "fill",
  colors
)

scale_colour_multi(
  ...,
  colours,
  values = NULL,
  na.value = "transparent",
  guide = "colourbar",
  aesthetics = "colour",
  colors
)
```

Arguments

`..., colours, values, na.value, guide, colors`	listed arguments in `scale_colour_gradientn` (e.g. `colours = list(c("white","red"),c("black","`
`aesthetics`	a `character` vector with names of aesthetic mapping.

Details

This function should only be called after all layers that this function affects are added to the plot.

The list elements of the listed arguments are assumed to follow the `aesthetics` order, i.e. the n*th* list element belongs to the n*th* aesthetic. When there are more list elements than n aesthetics, only

the first n*th* list elements are taken. When there are more `aesthetics` than list elements, the first list element is used for the remaining aesthethics.

In contrast to other `scale_*_continous`-family functions, the `guide` argument is interpreted before adding it to the plot instead of at the time of plot building. This behaviour ensures that the `available_aes` argument of the guides are set correctly, but may interfere with the `guides` function.

Value

A nested list-like structure of the class `MultiScale`.

Examples

```
# Setup dummy data
df <- rbind(data.frame(x = 1:3, y = 1, v = NA, w = 1:3, z = NA),
            data.frame(x = 1:3, y = 2, v = 1:3, w = NA, z = NA),
            data.frame(x = 1:3, y = 3, v = NA, w = NA, z = 1:3))

ggplot(df, aes(x, y)) +
  geom_raster(aes(fill1 = v)) +
  geom_raster(aes(fill2 = w)) +
  geom_raster(aes(fill3 = z)) +
  scale_fill_multi(aesthetics = c("fill1", "fill2", "fill3"),
                   colours = list(c("white", "red"),
                                  c("black", "blue"),
                                  c("grey50", "green")))
```

---

scale_listed *Add a list of scales for non-standard aesthetics*

Description

This function should only be called after all layers that the non-standard aesthetic scales affects have been added to the plot.

Inside a layer, the non-standard aesthetic should be part of the call to `aes` mapping.

May return a warning that the plot is ignoring unknown aesthetics.

Usage

```
scale_listed(scalelist, replaces = NULL)
```

Arguments

`scalelist`	A `list` wherein elements are the results of calls to a scale function with a non-standard aesthetic set as the `aesthetic` argument.
`replaces`	A `character` vector of the same length as- and parallel to- `scalelist`, indicating what standard aesthetic to replace with the non-standard aesthetic. Typically `"colour"` or `"fill"`.

Details

Distributes a list of non-standard aesthetics scales to the plot, substituting geom and scale settings as necessary to display the non-standard aesthetics. Useful for mapping different geoms to different scales for example.

Value

A `list` of which the elements are of the class `MultiScale`.

Examples

```
# Annotation of heatmap
iriscor <- cor(t(iris[, 1:4]))

df <- data.frame(
  x = as.vector(row(iriscor)),
  y = as.vector(col(iriscor)),
  value = as.vector(iriscor)
)

annotation <- data.frame(
  z = seq_len(nrow(iris)),
  Species = iris$Species,
  Leaves = ifelse(iris$Species == "setosa", "Short", "Long")
)

ggplot(df, aes(x, y)) +
  geom_raster(aes(fill = value)) +
  geom_tile(data = annotation,
            aes(x = z, y = -5, spec = Species), height = 5) +
  geom_tile(data = annotation,
            aes(y = z, x = -5, leav = Leaves), width = 5) +
  scale_listed(
    list(scale_fill_brewer(palette = "Set1", aesthetics = "spec"),
         scale_fill_brewer(palette = "Dark2", aesthetics = "leav")),
    replaces = c("fill", "fill")
  )
```

`stat_funxy` *Apply function to position coordinates*

Description

The function xy stat applies a function to the x- and y-coordinates of a layers positions by group. The `stat_centroid()` and `stat_midpoint()` functions are convenience wrappers for calculating centroids and midpoints. `stat_funxy()` by default leaves the data as-is, but can be supplied functions and arguments.

Usage

```
stat_funxy(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  funx = force,
  funy = force,
  argx = list(),
  argy = list(),
  crop_other = TRUE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_centroid(
  ...,
  funx = mean,
  funy = mean,
  argx = list(na.rm = TRUE),
  argy = list(na.rm = TRUE)
)

stat_midpoint(..., argx = list(na.rm = TRUE), argy = list(na.rm = TRUE))
```

Arguments

mapping	Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)).
geom	The geometric object to use display the data
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
...	Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.

funx, funy	A function to call on the layer's x and y positions respectively.
argx, argy	A named list containing arguments to the funx, and funy function calls.
crop_other	A logical of length one; wether the other data should be fitted to the length of x and y (default: TRUE). Useful to set to FALSE when funx or funy calculate summaries of length one that need to be recycled.
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. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders().

Details

This statistic only makes a minimal attempt at ensuring that the results from calling both functions are of equal length. Results of length 1 are recycled to match the longest length result.

Value

A StatFunxy ggproto object, that can be added to a plot.

Examples

```
p <- ggplot(iris, aes(Sepal.Width, Sepal.Length, colour = Species))

# Labelling group midpoints
p + geom_point() +
  stat_midpoint(aes(label = Species, group = Species),
                geom = "text", colour = "black")

# Drawing segments to centroids
p + geom_point() +
  stat_centroid(aes(xend = Sepal.Width, yend = Sepal.Length),
                geom = "segment", crop_other = FALSE)

# Drawing intervals
ggplot(iris, aes(Sepal.Width, Sepal.Length, colour = Species)) +
  geom_point() +
  stat_funxy(geom = "path",
             funx = median, funy = quantile,
             argy = list(probs = c(0.1, 0.9)))
```

---

stat_rle *Run length encoding*

Description

Run length encoding takes a vector of values and calculates the lengths of consequetive repeated values.

Usage

```
stat_rle(
  mapping = NULL,
  data = NULL,
  geom = "rect",
  position = "identity",
  ...,
  align = "none",
  na.rm = FALSE,
  orientation = "x",
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)).
geom	Use to override the default connection between geom_density() and stat_density().
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
...	Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.
align	A character of length one that effect the computed start and end variables. One of the following: "none" Take exact start and end x values. "center" Return start and end x values in between an end and the subsequent start. "start" Align start values with previous end values. "end" Align end values with next start values.
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
orientation	The orientation of the layer. The default (NA) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be

given explicitly by setting `orientation` to either `"x"` or `"y"`. See the *Orientation* section for more detail.

`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. It can also be a named logical vector to finely select the aesthetics to display.

`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 data is first ordered on the `x` aesthetic before run lengths are calculated for the `label` aesthetic.

Value

A `ggplot2` layer

Aesthetics

`stat_rle()` understands the following aesthetics (required aesthetics are in bold)

- **x**
- **label**
- group

Computed variables

start The `x` values at the start of every run.

end The `x` values at the end of every run.

start_id The index where a run starts.

end_id The index where a run ends.

run_id The index of a run.

runlength The length of a run.

runvalue The value associated with a run.

Examples

```
df <- data.frame(
  x = seq(0, 10, length.out = 100),
  y = sin(seq(0, 10, length.out = 100)*2)
)

# Label every run of increasing values
ggplot(df) +
  stat_rle(aes(x, label = diff(c(0, y)) > 0),
           align = "end") +
  geom_point(aes(x, y))
```

```
# Label every run above some threshold
ggplot(df) +
  stat_rle(aes(x, label = y > 0),
           align = "center") +
  geom_point(aes(x, y))

# Categorising runs, more complicated usage
ggplot(df) +
  stat_rle(aes(stage(x, after_stat = run_id),
               after_stat(runlength),
               label = cut(y, c(-1, -0.6, 0.6, 1)),
               fill = after_stat(runvalue)),
           geom = "col")
```

`stat_rollingkernel` *Rolling Kernel*

Description

A rolling kernel moves along one of the axes and assigns weights to datapoints depending on the distance to the kernel's location. It then calculates a weighted average on the y-values of the datapoints, creating a trendline. In contrast to (weighted) rolling averages, the interval between datapoints do not need to be constant.

Usage

```
stat_rollingkernel(
  mapping = NULL,
  data = NULL,
  geom = "line",
  position = "identity",
  ...,
  bw = "nrd",
  kernel = "gaussian",
  n = 256,
  expand = 0.1,
  na.rm = FALSE,
  orientation = "x",
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()` or `aes_()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

`data` The data to be displayed in this layer. There are three options:

If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.

A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.

A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x,10)`).

`geom` Use to override the default geom (`"line"`).

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

`...` Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

`bw` A bandwidth, which can be one of the following:

- A `numeric` of length one indicating a measure of kernel width, in data units.
- A `rel` object of length one constructed for setting a bandwidth relative to the group data range. Can be constructed with the `rel()` function.
- A `character` of length one, naming one of the functions documented in `bw.nrd()`.

`kernel` One of the following:

- A `function` that takes a vector of distances as first argument, a numeric bandwidth as second argument and returns relative weights.
- A `character` of length one that can take one of the following values:

  `"gaussian"` **or** `"norm"` A kernel that follows a normal distribution with 0 mean and bandwidth as standard deviation.

  `"mean"` **or** `"unif"` A kernel that follows a uniform distribution with $bandwidth * -0.5$ and $bandwidth * 0.5$ as minimum and maximum. This is similar to a simple, unweighted moving average.

  `"cauchy"` A kernel that follows a Cauchy distribution with 0 as location and bandwidth as scale parameters. The Cauchy distribution has fatter tails than the normal distribution.

`n` An `integer` of length one: how many points to return per group.

`expand` A `numeric` of length one: how much to expand the range for which the rolling kernel is calculated beyond the most extreme datapoints.

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

`orientation` A `character` of length one, either `"x"` (default) or `"y"`, setting the axis along which the rolling should occur.

`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. It can also be a named logical vector to finely select the aesthetics to display.

`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 *Layer* ggproto object.

Aesthetics

`stat_rollingkernel()` understands the following aesthetics (required aesthetics are in bold)

- **x**
- **y**
- group

Computed variables

`x` A sequence of ordered x positions.

`y` The weighted value of the rolling kernel.

`weight` The sum of weight strengths at a position.

`scaled` The fraction of weight strengths at a position. This is the same as `weight / sum(weight)` by group.

Examples

```
ggplot(mpg, aes(displ, hwy, colour = class)) +
  geom_point() +
  stat_rollingkernel()

# The (scaled) weights can be used to emphasise data-dense areas
ggplot(mpg, aes(displ, hwy, colour = class)) +
  geom_point() +
  stat_rollingkernel(aes(alpha = after_stat(scaled)))
```

`stat_theodensity` *Fitted theoretical density*

Description

Estimates the parameters of a given distribution and evaluates the probability density function with these parameters. This can be useful for comparing histograms or kernel density estimates against a theoretical distribution.

Usage

```
stat_theodensity(
  mapping = NULL,
  data = NULL,
  geom = "line",
  position = "identity",
  ...,
  distri = "norm",
  n = 512,
  fix.arg = NULL,
  start.arg = NULL,
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)).
geom	Use to override the default geom for stat_theodensity.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
...	Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.
distri	A character of length 1 naming a distribution without prefix. See details.
n	An integer of length 1 with the number of equally spaced points at which the density function is evaluated. Ignored if distribution is discrete.
fix.arg	An optional named list giving values of fixed parameters of the named distribution. Parameters with fixed value are not estimated by maximum likelihood procedures.
start.arg	A named list giving initial values of parameters for the named distribution. This argument may be omitted (default) for some distributions for which reasonable starting values are computed.

na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
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. It can also be a named logical vector to finely select the aesthetics to display.
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

Valid distri arguments are the names of distributions for which there exists a density function. The names should be given without a prefix (typically 'd', 'r', 'q' and 'r'). For example: "norm" for the normal distribution and "nbinom" for the negative binomial distribution. Take a look at distributions in the **stats** package for an overview.

There are a couple of distribution for which there exist no reasonable starting values, such as the Student t-distribution and the F-distribution. In these cases, it would probably be wise to provide reasonable starting values as a named list to the start.arg argument. When estimating a binomial distribution, it would be best to supply the size to the fix.arg argument.

By default, the y values are such that the integral of the distribution is 1, which scales well with the defaults of kernel density estimates. When comparing distributions with absolute count histograms, a sensible choice for aesthetic mapping would be aes(y = stat(count) * binwidth), wherein binwidth is matched with the bin width of the histogram.

For discrete distributions, the input data are expected to be integers, or doubles that can be divided by 1 without remainders.

Parameters are estimated using the fitdist() function in the **fitdistrplus** package using maximum likelihood estimation. Hypergeometric and multinomial distributions from the **stats** package are not supported.

Value

A *Layer* ggproto object.

Computed variables

density probability density

count density * number of observations - useful for comparing to histograms

scaled density scaled to a maximum of 1

See Also

Distributions fitdist geom_density geom_histogram

Examples

```
# A mixture of normal distributions where the standard deviation is
# inverse gamma distributed resembles a cauchy distribution.
x <- rnorm(2000, 10, 1/rgamma(2000, 2, 0.5))
df <- data.frame(x = x)

ggplot(df, aes(x)) +
  geom_histogram(binwidth = 0.1,
                 alpha = 0.3, position = "identity") +
  stat_theodensity(aes(y = stat(count) * 0.1, colour = "Normal"),
                   distri = "norm", geom = "line") +
  stat_theodensity(aes(y = stat(count) * 0.1, colour = "Cauchy"),
                   distri = "cauchy", geom = "line") +
  coord_cartesian(xlim = c(5, 15))

# A negative binomial can be understood as a Poisson-gamma mixture
df <- data.frame(x = c(rpois(500, 25),
                       rpois(500, rgamma(500, 5, 0.2))),
                 cat = rep(c("Poisson", "Poisson-gamma"), each = 500))

ggplot(df, aes(x)) +
  geom_histogram(binwidth = 1, aes(fill = cat),
                 alpha = 0.3, position = "identity") +
  stat_theodensity(aes(y = stat(count), colour = cat), distri = "nbinom",
                   geom = "step", position = position_nudge(x = -0.5)) +
  stat_summary(aes(y = x, colour = cat, x = 1),
               fun.data = function(x){data.frame(xintercept = mean(x))},
               geom = "vline")
```

`weave_factors` *Bind together factors*

Description

Computes a new factor out of combinations of input factors.

Usage

```
weave_factors(..., drop = TRUE, sep = ".", replaceNA = TRUE)
```

Arguments

`...`	The vectors
`drop`	A `logical` of length 1 which when `TRUE` will remove combinations of factors not occurring in the input data.
`sep`	A `character` of length 1 with a string to delimit the new level labels.
`replaceNA`	A `logical` of length 1: replace `NA` values with empty strings?

Details

`weave_factors()` broadly resembles `interaction(...,lex.order = TRUE)`, with a slightly altered approach to non-factor inputs. In other words, this function orders the new levels such that the levels of the first input variable in `...` is given priority over the second input, the second input has priority over the third, etc.

This function treats non-factor inputs as if their levels were `unique(as.character(x))`, wherein `x` represents an input.

Value

A `factor` representing combinations of input factors.

See Also

`interaction`

Examples

```
f1 <- c("banana", "apple", "apple", "kiwi")
f2 <- factor(c(1, 1:3), labels = c("house", "cat", "dog"))

# Notice the difference in level ordering between the following:
interaction(f1, f2, drop = TRUE, lex.order = TRUE)
interaction(f1, f2, drop = TRUE, lex.order = FALSE)
weave_factors(f1, f2)

# The difference is in how characters are interpreted
# The following are equivalent
interaction(f1, f2, drop = TRUE, lex.order = TRUE)
weave_factors(as.factor(f1), f2)
```

Index