Package: gg.layers (via r-universe)

May 28, 2026

Title ggplot layers

Version 0.1.3

Description What the package does (one paragraph).

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 4.1)

Imports dplyr, magrittr, stringr, rlang, grid, ggplot2 (>= 3.5.0), gtable, data.table, gggrid, gridtext, ggtext, ggpp, glue, broom, ggh4x, ggpattern, ggplotify, rtrend, methods

Suggests testthat (>= 3.0.0), rcolors, scales, sf, stars, raster, mapview, lattice, cowplot, gridExtra, showtext, knitr, rmarkdown

Config/testthat/edition 3

VignetteBuilder knitr

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/pak/sysreqs libabsl-dev cmake libfftw3-dev libgdal-dev gdal-bin libgeos-dev make libicu-dev libjpeg-dev libpng-dev libuv1-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

Repository https://rpkgs.r-universe.dev

Date/Publication 2026-04-28 10:14:07 UTC

RemoteUrl https://github.com/rpkgs/gg.layers

RemoteRef HEAD

RemoteSha 9f565e69aa73aa1945e495086643624390e5e65a

Contents

`add_colorbar` *add_colorbar*

Description

add_colorbar

Usage

```
add_colorbar(
  p,
  g,
  width = NULL,
  height = NULL,
  title = NULL,
  space = "right",
  legend.title = element_text(hjust = 0, vjust = 0, size = 14, family = "Times")
)
```

Arguments

p	a ggplot object
g	a grob object, colorbar
width, height	width and height of the colorbar
title	all title elements: plot, axes, legends (element_text(); inherits from text)
space	one of c("left", "bottom")
legend.title	title of legend (element_text(); inherits from title)

---

add_grob *add grob to a plot*

Description

add grob to a plot

Usage

```
add_grob(p, ..., ggplot = TRUE)
```

Arguments

p	ggplot object
...	grob objects
ggplot	logical, if TRUE, return a ggplot object

Examples

```
library(ggplot2)
library(grid)

p <- ggplot(mtcars, aes(mpg, disp)) +
  geom_point() +
  facet_wrap(~cyl)
```

```
g1 <- textGrob("hello",
  x = 0.98, y = 0.1, hjust = 1, vjust = 0,
  gp = gpar(fontfamily = "Times"))
g2 <- element_grob_text(element_text(family = "Times", hjust = 1, vjust = 0, size = 12),
  label = "Hello world", x = 0.98, y = 0.2)

add_grob(p, g1, g2)
add_grob(p, g1) %>% add_grob(g2)
```

`colourbar_triangle` *colourbar_triangle*

Description

colourbar_triangle

Usage

```
colourbar_triangle(...)
```

Arguments

...	parameters passed to ggplot2::guide_colourbar

References

1. https://stackoverflow.com/questions/68440366/how-can-i-add-triangles-to-a-ggplot2-colorba

Examples

```
library(ggplot2)

g <- ggplot(mtcars, aes(mpg, wt)) +
  geom_point(aes(colour = drat))

g + scale_colour_viridis_c(
  limits = c(3, 5), oob = scales::oob_squish,
  guide = colourbar_triangle()
)
```

`draw_polygon` *draw horizontal and vertical polygons*

Description

draw horizontal and vertical polygons

Usage

```
draw_polygon(
  vals,
  x = NULL,
  type = "horizontal",
  length.out = 10000,
  col.regions = c("blue", "red"),
  alpha = 0.6,
  zlim = c(-Inf, Inf),
  ...
)
```

Arguments

`vals`	numeric vector
`x`	The corresponding x position of `vals`
`type`	one of ”horizontal” or ”vertical”
`length.out`	the length of interpolated `vals` and `x`
`col.regions`	A vector of colors, or a function to produce a vecor of colors, to be used if `region=TRUE`. Each interval defined by `at` is assigned a color, so the number of colors actually used is one less than the length of `at`. See `level.colors` for details on how the color assignment is done.
`alpha`	the alpha of polygon’s fill color
`zlim`	limits of `vals`
`...`	Extra parameters.

Examples

```
set.seed(1)
y <- rnorm(10)
x <- seq_along(y)
```

`geom_annotation` *geom_annotation*

Description

geom_annotation

Usage

```
geom_annotation(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  xmin = 0,
  xmax = 0.5,
  ymin = 0,
  ymax = 0.5,
  just = c(0, 0),
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = FALSE
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `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`	A tibble with the column of `grob`
`stat`	The statistical transformation to use on the data for this layer. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following: • A `Stat` ggproto subclass, for example `StatCount`. • A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`. • For more information and other ways to specify the stat, see the layer stat documentation.
`position`	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following: • The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.

- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`xmin`, `xmax` x location (in data coordinates) giving horizontal location of raster

`ymin`, `ymax` y location (in data coordinates) giving vertical location of raster

`just` A string or numeric vector specifying the justification of the viewport relative to its (x, y) location. If there are two values, the first value specifies horizontal justification and the second value specifies vertical justification. Possible string values are: `"left"`, `"right"`, `"centre"`, `"center"`, `"bottom"`, and `"top"`. For numeric values, 0 means left alignment and 1 means right alignment.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`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. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`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. `annotation_borders()`.

Examples

```
## make plot
library(ggplot2)
d <- data.frame(x = 1:3, y = 1:3,
                varname = c("T_min", "T_max", "T_avg"))
p = ggplot(d, aes(x, y)) +
  geom_point() +
  geom_richtext(aes(label = varname), x = 2, y = 2) +
  facet_wrap(~varname, labeller = label_mk) +
  theme(
    strip.text = element_markdown(face = "bold", margin = margin(t = 1, b = 0))
  )

## make legend
brks = 1:5 %>% c(-Inf, ., Inf)
nbrk <- length(brks) - 1
cols <- rcolors::get_color("amwg256", nbrk)
g = make_colorbar(brks, col = cols, space = "right")
d_lgd = tibble(varname = d$varname, grob = rep(list(g), 3))

## add legend to each panel
p2 = p + geom_annotation(data = d_lgd, aes(grob = grob), xmin = 0.8, xmax = 1, ymax = 1)
p2
```

---

geom_annotation_func *geom_annotation*

Description

geom_annotation

Usage

```
geom_annotation_func(
  mapping = NULL,
  data = NULL,
  plot.fun = NULL,
  ...,
  x = 0,
  y = 0,
  width = unit(0.5, "npc"),
  height = unit(0.5, "npc"),
  just = c(0, 0)
)
```

Arguments

mapping	Set of aesthetic mappings created by 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	A tibble with the column of grob
plot.fun	function to plot, p <- plot.fun(data, ...)
...	other parameters to plot.fun
x	A numeric vector or unit object specifying x-location.
y	A numeric vector or unit object specifying y-location.
width	A numeric vector or unit object specifying width.
height	A numeric vector or unit object specifying height.
just	A string or numeric vector specifying the justification of the viewport relative to its (x, y) location. If there are two values, the first value specifies horizontal justification and the second value specifies vertical justification. Possible string values are: "left", "right", "centre", "center", "bottom", and "top". For numeric values, 0 means left alignment and 1 means right alignment.

Examples

```
## prepare data
library(gg.layers)
library(ggplot2)
library(rcolors)

data("d_trendPerc")
d_mask <- mutate(d_trendPerc, mask = perc <= 0.99) %>% as_tibble()
n <- nrow(d_mask) * 2
dat <- rbind(cbind(type = "a", d_mask), cbind(type = "b", d_mask)) %>%
  mutate(val = rnorm(n))

brks <- seq(0.9, 1, 0.025)
nbrk <- length(brks) - 1
cols <- get_color(rcolors$amwg256, nbrk)

## option1
# the part of not significant
ggplot(data = dat, aes(x, y)) +
  geom_raster(aes(fill = perc)) +
  layer_barchart(aes(z = val),
    width = unit(0.3, "npc"),
    height = unit(0.3, "npc"),
    brks = brks, cols = cols
  ) +
  facet_wrap(~type)

## option2
func <- function(data, ...) {
```

```
  add_barchart(data$z, brks, cols, ...)
}

ggplot(data = dat, aes(x, y)) +
  geom_raster(aes(fill = perc)) +
  geom_annotation_func(aes(z = val), plot.fun = func) +
  facet_wrap(~type)
```

`geom_boxplot2` *A box and whiskers plot (in the style of Tukey)*

Description

The boxplot compactly displays the distribution of a continuous variable. It visualises five summary statistics (the median, two hinges and two whiskers), and all "outlying" points individually.

Usage

```
geom_boxplot2(
  mapping = NULL,
  data = NULL,
  stat = "boxplot",
  position = "dodge2",
  ...,
  outlier.colour = NULL,
  outlier.color = NULL,
  outlier.fill = NULL,
  outlier.shape = 19,
  outlier.size = 1.5,
  outlier.stroke = 0.5,
  outlier.alpha = NULL,
  show.errorbar = TRUE,
  width.errorbar = 0.7,
  notch = FALSE,
  notchwidth = 0.5,
  varwidth = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), 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

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `...`. Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`outlier.colour`, `outlier.color`, `outlier.fill`, `outlier.shape`, `outlier.size`, `outlier.stroke`, `outlier.alpha`
: Default aesthetics for outliers. Set to `NULL` to inherit from the data's aesthetics.

`show.errorbar`
: whether to show errorbar (default TRUE)

`width.errorbar`
: the width of errorbar (default 0.7)

`notch`
: If `FALSE` (default) make a standard box plot. If `TRUE`, make a notched box plot. Notches are used to compare groups; if the notches of two boxes do not overlap, this suggests that the medians are significantly different.

`notchwidth`
: For a notched box plot, width of the notch relative to the body (defaults to `notchwidth = 0.5`).

`varwidth`
: If `FALSE` (default) make a standard box plot. If `TRUE`, boxes are drawn with widths proportional to the square-roots of the number of observations in the groups (possibly weighted, using the `weight` aesthetic).

`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. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`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. `annotation_borders()`.

Aesthetics

`geom_boxplot()` understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- `x` ***or*** `y`
- `lower` ***or*** `xlower`
- `upper` ***or*** `xupper`
- `middle` ***or*** `xmiddle`
- `ymin` ***or*** `xmin`
- `ymax` ***or*** `xmax`
- `alpha` → `NA`
- `colour` → via `theme()`
- `fill` → via `theme()`
- `group` → inferred
- `linetype` → via `theme()`
- `linewidth` → via `theme()`
- `shape` → via `theme()`

- size → via `theme()`
- `weight` → `1`
- `width` → `0.9`

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

Summary statistics

The lower and upper hinges correspond to the first and third quartiles (the 25th and 75th percentiles). This differs slightly from the method used by the `boxplot()` function, and may be apparent with small samples. See `boxplot.stats()` for more information on how hinge positions are calculated for `boxplot()`.

The upper whisker extends from the hinge to the largest value no further than 1.5 * IQR from the hinge (where IQR is the inter-quartile range, or distance between the first and third quartiles). The lower whisker extends from the hinge to the smallest value at most 1.5 * IQR of the hinge. Data beyond the end of the whiskers are called "outlying" points and are plotted individually.

In a notched box plot, the notches extend `1.58 * IQR / sqrt(n)`. This gives a roughly 95% confidence interval for comparing medians. See McGill et al. (1978) for more details.

References

1. McGill, R., Tukey, J. W. and Larsen, W. A. (1978) Variations of box plots. The American Statistician 32, 12-16.

See Also

`ggplot2::geom_quantile()` for continuous `x`, `ggplot2::geom_violin()` for a richer display of the distribution, and `ggplot2::geom_jitter()` for a useful technique for small data.

Examples

```
library(ggplot2)

p <- ggplot(mpg, aes(class, hwy))
p + geom_boxplot2()
p + geom_boxplot2(width.errorbar = 0.5)
p + geom_boxplot2(width = 0.5)
# Orientation follows the discrete axis
# ggplot(mpg, aes(hwy, class)) + geom_boxplot2()

p + geom_boxplot2(notch = TRUE)
p + geom_boxplot2(varwidth = TRUE)
p + geom_boxplot2(fill = "white", colour = "#3366FF")
# By default, outlier points match the colour of the box. Use
# outlier.colour to override
p + geom_boxplot2(outlier.colour = "red", outlier.shape = 1)
# Remove outliers when overlaying boxplot with original data points
```

```
p + geom_boxplot2(outlier.shape = NA) + geom_jitter(width = 0.2)

# Boxplots are automatically dodged when any aesthetic is a factor
p + geom_boxplot2(aes(colour = drv))

# You can also use boxplots with continuous x, as long as you supply
# a grouping variable. cut_width is particularly useful
# ggplot(diamonds, aes(carat, price)) +
#   geom_boxplot2()
# ggplot(diamonds, aes(carat, price)) +
#   geom_boxplot2(aes(group = cut_width(carat, 0.25)))
# Adjust the transparency of outliers using outlier.alpha
# ggplot(diamonds, aes(carat, price)) +
#   geom_boxplot2(aes(group = cut_width(carat, 0.25)), outlier.alpha = 0.1)

# It's possible to draw a boxplot with your own computations if you
# use stat = "identity":
y <- rnorm(100)
df <- data.frame(
  x = 1,
  y0 = min(y),
  y25 = quantile(y, 0.25),
  y50 = median(y),
  y75 = quantile(y, 0.75),
  y100 = max(y)
)
ggplot(df, aes(x)) +
  geom_boxplot2(
   aes(ymin = y0, lower = y25, middle = y50, upper = y75, ymax = y100),
   stat = "identity"
 )
```

---

geom_hspan *geom_hspan*

Description

- xmin, xmax
- ymin, ymax: optional

Usage

```
geom_hspan(
  mapping = NULL,
  data = NULL,
  stat = "hspan",
  position = "identity",
  ...,
  ymin = -Inf,
```

```
  ymax = Inf,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), 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. When using a geom_*() function to construct a layer, the stat argument can be used to override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`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. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`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. `annotation_borders()`.

Examples

```
library(ggplot2)
library(data.table)

d = data.table(x = 1:10, y = 1:10)
d_span = data.table(xmin = c(1, 4), xmax = c(3, 5), group = 1:2)

ggplot(d, aes(x, y)) +
  geom_point() +
  geom_hspan(data = d_span, aes(x = NULL, y = NULL, xmin = xmin, xmax = xmax, group = group),
    alpha = 0.2, fill = "yellow")
```

`geom_latFreq` *geom_latFreq*

Description

geom_latFreq

Usage

```
geom_latFreq(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  options = list(),
  bbox = c(185, 240, -60, 90),
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), 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. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following: • A `Stat` ggproto subclass, for example `StatCount`. • A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`. • For more information and other ways to specify the stat, see the layer stat documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`options` parameters of `make_latFreq()`

`bbox` bounding box of the plot, in the form of `c(xmin, xmax, ymin, ymax)`.

`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. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`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. `annotation_borders()`.

Examples

```
library(ggplot2)
# the lattice version: https://github.com/CUG-hydro/Liu2021-JGRA-ET_trends_attribution

# f <- "data-raw/AridityIndex_MSWEP-prcp_div_GLEAM-Ep_1980-2020.tif"
# r <- terra::rast(f)
# d <- as.data.table(r, xy = TRUE) |> set_names(c("x", "y", "z"))
d = AridityIndex

# make_latFreq(d$y, d$z, debug = TRUE, zlim = c(-2, 2), is_spatial = TRUE)
# brks <- c(-Inf, 0.05, 0.2, 0.5, 0.65, Inf)
brks <- c(-Inf, 0.05, 0.2, 0.5, 0.65, 1:5, 20, Inf)

nbrk <- length(brks) - 1
cols <- rcolors::get_color("amwg256", nbrk) |> rev()

p = ggplot(d, aes(x, y, z = z)) +
  geom_raster_filled(breaks = brks) +
  geom_latFreq(options = list(is_spatial = TRUE, zlim = c(-1, 1)*10),
    bbox = c(190, 240, -60, 90)) +
  coord_cartesian(xlim = c(-180, 240), ylim = c(-60, 90), expand = FALSE, clip = "on") +
  scale_x_continuous(limits = c(-180, 240), breaks = seq(-180, 180, 60)) +
  scale_fill_manual(values =cols) +
  labs(x = NULL, y = NULL)
p
```

`geom_mk` *geom_mk*

Description

geom_mk

Usage

```
geom_mk(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  fun_slope = slope_mk,
  na.rm = FALSE,
```

```
  show.legend = NA,
  inherit.aes = TRUE
)

stat_mk(
  mapping = NULL,
  data = NULL,
  geom = "abline",
  position = "identity",
  fun_slope = slope_mk,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)

stat_spike(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  halfwin = 3,
  sd.times = 3,
  trs = NA,
  verbose = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`.
`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. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following: • A `Stat` ggproto subclass, for example `StatCount`.

- A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`.
- For more information and other ways to specify the stat, see the layer stat documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`fun_slope` function to calculate slope, default `rtrend::slope_mk()`

`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. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`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. `annotation_borders()`.

Value

No return. This function is used to calculate data for gglot2 `geom_*`, just like `ggplot2::stat_smooth()`.

No return. This function is used to calculate data for gglot2 `geom_*`, just like `ggplot2::stat_smooth()`.

Examples

```
library(ggplot2)
library(rtrend)

ggplot(mpg, aes(displ, hwy, colour = drv)) +
  geom_point() +
  stat_mk(linewidth = 1, fun_slope = slope_mk) +
  geom_smooth(method = "lm", se = FALSE, linetype = 2)

ggplot(mpg, aes(displ, hwy, colour = drv)) +
  geom_point() +
  geom_mk(linewidth = 1, fun_slope = slope_mk) +
  geom_smooth(method = "lm", se = FALSE, linetype = 2)
library(ggplot2)
library(rtrend)

ggplot(mpg, aes(displ, hwy, colour = drv)) +
  geom_point() +
  stat_mk(linewidth = 1, fun_slope = slope_mk) +
  geom_smooth(method = "lm", se = FALSE, linetype = 2)

ggplot(mpg, aes(displ, hwy, colour = drv)) +
  geom_point() +
  geom_mk(linewidth = 1, fun_slope = slope_mk) +
  geom_smooth(method = "lm", se = FALSE, linetype = 2)
```

`geom_movmean` *geom_movmean*

Description

geom_movmean

Usage

```
geom_movmean(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  halfwin = 3,
  show.diff = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), 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. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following: • A `Stat` ggproto subclass, for example `StatCount`. • A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`. • For more information and other ways to specify the stat, see the layer stat documentation.
`position`	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following: • The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.

- For more information and other ways to specify the position, see the layer position documentation.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`halfwin` halfwin of movmean, `rtrend::movmean()`

`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. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`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. `annotation_borders()`.

Examples

```
library(data.table)
library(ggplot2)
data("GPP_US_MMS")
dat = melt(GPP_US_MMS[, .(date, SM, GPP)], "date")
```

```
ggplot(dat, aes(date, value)) +
  geom_line() +
  geom_movmean(halfwin = 5, linewidth = 1, alpha = 0.7, color = "pink") +
  # geom_movmean(halfwin = 5, show.diff=TRUE, linewidth = 0.2, alpha = 0.7, color = "red") +
  geom_spike(halfwin = 2, sd.times = 6, color = "red", verbose = TRUE) +
  # geom_spike(halfwin = 5, trs = 3, color = "blue") +
  facet_wrap(~variable, scales = "free_y")
```

`geom_prcpRunoff` *Draw precipitation bar on the top of the panel*

Description

Draw precipitation bar on the top of the panel

Usage

```
geom_prcpRunoff(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  linejoin = "mitre",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  prcp.ratio = 0.5,
  params_prcp = list(),
  prcp.coef = 1,
  qmax = NULL,
  sec.axis = NULL,
  sec.name = "Precipitation (mm)"
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `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. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following:

- A `Stat` ggproto subclass, for example `StatCount`.
- A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`.
- For more information and other ways to specify the stat, see the layer stat documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.

- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`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. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`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. `annotation_borders()`.

`prcp.ratio` the ratio of precipitation to `ymax` (not used currently)

`params_prcp`
- `color`: color of precipitation
- `fill`: fill of precipitation

`prcp.coef` coefficient of precipitation, `y_new = qmax - prcp * prcp.coef` prcp.coef = qmax / max(prcp)

`qmax` maximum of streamflow, used to calculate `prcp.coef`

`sec.axis` secondary axis for precipitation, returned by `ggplot2::sec_axis()`

`sec.name` name of secondary axis

Aesthetics

- `x`: date or continuous variable
- `y`: runoff
- `prcp`: precipitation

Author(s)

Xie YuXuan and Dongdong Kong

Examples

```
library(ggplot2)

col_prcp = "blue"  #"#3e89be"
col_runoff = "black"  # "darkorange"

my_theme <-
  theme_dual_axis(col_runoff, col_prcp) +
  theme(
    legend.position.inside = c(0, 1),
```

```
    legend.justification = c(0, 1),
    legend.background = element_blank(),
    legend.key = element_blank(),
    # axis.ticks = element_blank(),
    axis.text = element_text(color = "black"),
    axis.text.x = element_text(angle = 60, hjust = 1),
    strip.background = element_blank(),
    strip.text = element_text(face = "bold", hjust = 0)
  )

## Visualization ---------------------------------------------------------------
dat <- runoff_data
qmax <- max(dat$Q) * 1.1
prcp.coef <- guess_prcp_coef(qmax, dat$prcp, ratio = 0.5)
# prcp.coef = qmax / pmax * ratio

ggplot(dat, aes(x = time, Q)) +
  # theme_test() +
  geom_line() +
  geom_prcpRunoff(
    aes(prcp = prcp, color = flood_type),
    params_prcp = list(color = col_prcp, fill = col_prcp),
    prcp.coef = prcp.coef,
    qmax = qmax,
    color = col_runoff, linewidth = 0.5
  ) +
  facet_wrap(~flood_type, scales = "free") +
  # scale_y_precipitation(sec.name = "Precipitation (mm)", coef = set_coef) +
  scale_x_datetime(date_labels = "%m/%d") +
  my_theme +
  labs(x = "Date", y = expression("Streamflow (m"^"3" * "/s)"))
```

`geom_raster_filled` *geom_raster_filled*

Description

geom_raster_filled

Usage

```
geom_raster_filled(
  mapping = NULL,
  data = NULL,
  stat = "raster_filled",
  position = "identity",
  ...,
  breaks = NULL,
  hjust = 0.5,
  vjust = 0.5,
```

```
  interpolate = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_raster_filled(
  mapping = NULL,
  data = NULL,
  geom = "raster",
  position = "identity",
  ...,
  breaks = NULL,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), 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. When using a geom_*() function to construct a layer, the stat argument can be used to override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position.

- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`breaks` One of:

- Numeric vector to set the contour breaks
- A function that takes the range of the data and binwidth as input and returns breaks as output. A function can be created from a formula (e.g. ~ fullseq(.x, .y)).

Overrides `binwidth` and `bins`. By default, this is a vector of length ten with `pretty()` breaks.

`hjust`, `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.

`interpolate` If `TRUE` interpolate linearly, if `FALSE` (the default) don't interpolate.

`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. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`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. `annotation_borders()`.

`geom` The geometric object to use to display the data for this layer. When using a `stat_*()` function to construct a layer, the `geom` argument can be used to override the default coupling between stats and geoms. The `geom` argument accepts the following:

- A `Geom` ggproto subclass, for example `GeomPoint`.
- A string naming the geom. To give the geom as a string, strip the function name of the `geom_` prefix. For example, to use `geom_point()`, give the geom as `"point"`.
- For more information and other ways to specify the geom, see the layer geom documentation.

`geom_richtext_npc` *Text with Normalised Parent Coordinates*

Description

Text with Normalised Parent Coordinates

Usage

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

annotate_richtext_npc(x, y, label, size = 5, family = "", ...)
```

```
annotate_richlabel_npc(
  x,
  y,
  label,
  size = 5,
  family = "",
  fill = "white",
  label.color = "black",
  ...
)

annotate_label_npc(
  x,
  y,
  label,
  size = 5,
  family = "",
  fill = "white",
  label.color = "black",
  ...
)
```

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 jointy 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.
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`.
`label.padding`	Amount of padding around label. Defaults to 0.25 lines.
`label.margin`	Unit vector of length four specifying the margin outside the text label.
`label.r`	Radius of rounded corners. Defaults to 0.15 lines.
`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()`.
`x`	A numeric vector or unit object specifying x-values.
`y`	A numeric vector or unit object specifying y-values.
`label`	A character or expression vector. Other objects are coerced by `as.graphicsAnnot`.

Aesthetics

`geom_richtext()` understands the following aesthetics (required aesthetics are in bold; select aesthetics are annotated):

- `x`
- `y`
- `label`
- `alpha`
- `angle`
- `colour` Default color of label text and label outline.
- `family`
- `fontface`
- `fill` Default fill color of label background.
- `group`
- `hjust`
- `label.colour` Color of label outline. Overrides `colour`.
- `label.size` Width of label outline.
- `lineheight`
- `size` Default font size of label text.
- `text.colour` Color of label text. Overrides `colour`.
- `vjust`

See Also

ggtext::geom_richtext(), geom_richtext2()

Examples

```
library(ggplot2)

## first example
labels <- c(
  "gC m^{-2} d^{-1}",
  "gC m^-2 d^-1",
  "gC m_{-2} d_{-1}",
  "gC m_-2 d_-1"
  # "gC \n mm/d"
)
x = 0.2
y = seq_along(labels)/10

ggplot() + annotate_richtext_npc(x, y, labels, size = 5)
ggplot() + annotate_richlabel_npc(x, y, labels, size = 5, label.color = "red")

# Another option
d = data.frame(x = 0.2, y = seq_along(labels)/10, label = labels)
ggplot(d, aes(npcx = x, npcy = y)) +
  geom_richtext_npc(aes(npcx = x, npcy = y, label = label))

# remove fill and label.color
ggplot(d, aes(npcx = x, npcy = y)) +
  geom_richtext_npc(aes(npcx = x, npcy = y, label = label),
                    fill = "white", label.color = "red")

## second example
d$label <- c(
  "Some text **in bold.**",
  "Linebreaks<br>Linebreaks<br>Linebreaks",
  "*x*<sup>2</sup> + 5*x* + *C*<sub>*i*</sub>",
  "Some <span style='color:blue'>blue text **in bold.**</span><br>And
  *italics text.*<br>
  And some <span style='font-size:18pt; color:black'>large</span> text."
)
ggplot(d, aes(npcx = x, npcy = y)) +
  geom_richtext_npc(aes(npcx = x, npcy = y, label = label))

## test for `str_mk`
library(magrittr)
indexes_lev = c("DOY_first", "DOY_last", "HWD", "HWI", "HWS_mean",
  "HWS_sum", "HWA_avg", "HWA_max", "HWA_sum")
labels = indexes_lev %>% str_mk() #%>% label_tag(expression = F)
d = data.frame(x = 0.5, y = 0.5, label = labels)

ggplot(d) +
  facet_wrap(~label) +
```

```
  theme(strip.text.x = element_textbox(face = "bold")) +
  geom_richtext_npc(aes(npcx = x, npcy = y, label = label))
```

`geom_richtext2` *Richtext labels*

Description

This geom draws text labels similar to `ggplot2::geom_label()`, but formatted using basic markdown/html. Parameter and aesthetic names follow the conventions of `ggplot2::geom_label()`, and therefore the appearance of the frame around the label is controlled with `label.colour`, `label.padding`, `label.margin`, `label.size`, `label.r`, even though the same parameters are called `box.colour`, `box.padding`, `box.margin`, `box.size`, and `box.r` in `ggtext::geom_textbox()`. Most styling parameters can be used as aesthetics and can be applied separately to each text label drawn. The exception is styling parameters that are specified as grid units (e.g., `label.padding` or `label.r`), which can only be specified for all text labels at once. See examples for details.

Usage

```
geom_richtext2(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  nudge_x = 0,
  nudge_y = 0,
  label.padding = unit(c(0.25, 0.25, 0.25, 0.25), "lines"),
  label.margin = unit(c(0, 0, 0, 0), "lines"),
  label.r = unit(0.15, "lines"),
  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 jointy 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.
`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`.
`label.padding`	Amount of padding around label. Defaults to 0.25 lines.
`label.margin`	Unit vector of length four specifying the margin outside the text label.
`label.r`	Radius of rounded corners. Defaults to 0.15 lines.
`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()`.

Value

A ggplot2 layer that can be added to a plot created with `ggplot2::ggplot()`.

Aesthetics

`geom_richtext()` understands the following aesthetics (required aesthetics are in bold; select aesthetics are annotated):

- `x`
- `y`
- `label`
- `alpha`
- `angle`
- `colour` Default color of label text and label outline.

- `family`
- `fontface`
- `fill` Default fill color of label background.
- `group`
- `hjust`
- `label.colour` Color of label outline. Overrides `colour`.
- `label.size` Width of label outline.
- `lineheight`
- `size` Default font size of label text.
- `text.colour` Color of label text. Overrides `colour`.
- `vjust`

See Also

`ggtext::geom_textbox()`, `ggtext::element_markdown()`

Examples

```
library(ggplot2)

labels1 = c(
  "Some text **in bold.**",
  "Linebreaks<br>Linebreaks<br>Linebreaks",
  "*x*<sup>2</sup> + 5*x* + *C*<sub>*i*</sub>",
  "Some <span style='color:blue'>blue text **in bold.**</span><br>And *italics text.*<br>
    And some <span style='font-size:18pt; color:black'>large</span> text."
)

labels2 <- c(
  "gC m^{-2} d^{-1}",
  "gC m^-2 d^-1",
  "gC m_{-2} d_{-1}",
  "gC m_-2 d_-1"
  # "gC \n mm/d"
)

df <- data.frame(
  x = c(.2, .1, .5, .9),
  y = c(.8, .4, .1, .5),
  hjust = c(0.5, 0, 0, 1),
  vjust = c(0.5, 1, 0, 0.5),
  angle = c(0, 0, 45, -45),
  color = c("black", "blue", "black", "red"),
  fill = c("cornsilk", "white", "lightblue1", "white")
)

fun <- function(labels) {
  df$label = labels
  ggplot(df, aes(x, y, label = label, angle = angle, color = color,
```

```
                hjust = hjust, vjust = vjust)) +
    geom_richtext2(
      fill = NA, label.color = NA, # remove background and outline
      label.padding = grid::unit(rep(0, 4), "pt") # remove padding
    ) +
    geom_point(color = "black", size = 2) +
    scale_color_identity() +
    xlim(0, 1) +
    ylim(0, 1)
}
fun(labels1)
fun(labels2)
# labels without frame or background are also possible
```

`geom_spike` *geom_spike*

Description

geom_spike

Usage

```
geom_spike(
  mapping = NULL,
  data = NULL,
  stat = "spike",
  position = "identity",
  ...,
  halfwin = 3,
  sd.times = 3,
  trs = NA,
  verbose = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), 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. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following:

- A `Stat` ggproto subclass, for example `StatCount`.
- A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`.
- For more information and other ways to specify the stat, see the layer stat documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density",`

`adjust = 0.5)`. The stat's documentation lists which parameters it can accept.

- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`halfwin` halfwin of movmean, `rtrend::movmean()`

`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. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`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. `annotation_borders()`.

Examples

```
library(data.table)
library(ggplot2)
data("GPP_US_MMS")
dat = melt(GPP_US_MMS[, .(date, SM, GPP)], "date")

ggplot(dat, aes(date, value)) +
  geom_line() +
  geom_movmean(halfwin = 5, linewidth = 1, alpha = 0.7, color = "pink") +
  # geom_movmean(halfwin = 5, show.diff=TRUE, linewidth = 0.2, alpha = 0.7, color = "red") +
  geom_spike(halfwin = 2, sd.times = 6, color = "red", verbose = TRUE) +
  # geom_spike(halfwin = 5, trs = 3, color = "blue") +
  facet_wrap(~variable, scales = "free_y")
```

`geom_taylor` *geom_taylor*

Description

geom_taylor

Usage

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

```
  position = "identity",
  ...,
  obs.colour = "black",
  obs.size = 5,
  show.obs.label = TRUE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), 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. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following: • A `Stat` ggproto subclass, for example `StatCount`. • A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`. • For more information and other ways to specify the stat, see the layer stat documentation.
`position`	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following: • The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`. • For more information and other ways to specify the position, see the layer position documentation.
`...`	Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further

arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`obs.colour` color of observed point.

`obs.size` size of observed point.

`show.obs.label` logical, whether to show the label of observed point.

`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. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`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. `annotation_borders()`.

Examples

```
library(ggplot2)
library(dplyr)

data = dummy_model %>%
  group_by(model, variable) %>%
```

```
  group_modify(~taylor_data(.$obs, .$mod)) #%>% as.data.table()
print(data)

mar = 0.01
p = ggplot(data) +
  geom_taylor(aes2(sd.obs, sd.mod, R, color = model), obs.colour = "black", obs.size = 5) +
  # geom_point(aes(color = model), size = 5) +
  facet_wrap(~variable) +
  labs(color = NULL) +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    legend.position = c(1, 1) - c(1, 1/3)*mar,
    legend.justification = c(1, 1)
  )
p
# p = last_plot()
# Ipaper::write_fig(p, "Rplot.pdf")
```

`ggplot_legend` *ggplot_legend*

Description

ggplot_legend

Usage

```
ggplot_legend(g)
```

Arguments

g	A ggplot object

Value

A grob object

`ggplot_multiaxis` *ggplot_multiaxis*

Description

ggplot_multiaxis

Usage

```
ggplot_multiaxis(..., linewidth = 1.4, tck = 0.2, x = -0.02)

ggplot_multiAxis(..., linewidth = 1.4, tck = 0.2, x = -0.02)
```

Arguments

...	ggplot2 objects, which should have axis.tick.y.right and axis.title.y.left
linewidth	line width of right axis line
tck	tick length of right axis

Examples

```
library(ggplot2)
library(rlang)
library(gg.layers)

plot_1var <- function(d, var, color = "black", lwd = 0.4, alpha = 0.6) {
  p <- ggplot(d, aes(date, !!sym(var))) +
    geom_line(color = color, linewidth = lwd, alpha = alpha) +
    # facet_wrap(~site) +
    scale_x_date(date_breaks = "1 year", date_labels = "%Y") +
    theme(
      panel.background = element_rect(fill = "transparent", color = "black"),
      plot.margin = margin(r = 5, l = 5, t = 2, b = 2),
      axis.ticks.y.left = element_line(color = color),
      axis.text.y.left = element_text(color = color),
      axis.title.y.left = element_text(color = color),
      axis.ticks.y.right = element_line(color = color),
      axis.text.y.right = element_text(color = color),
      axis.title.y.right = element_text(color = color, margin = margin(l = 2, r = 5)),
      panel.grid.major = element_blank(), # get rid of major grid
      panel.grid.minor = element_blank()
    ) # get rid of minor grid
  p
}

# GPP_US_MMS = d[year(date) >= 2013]
# usethis::use_data(GPP_US_MMS)
p_gpp <- plot_1var(GPP_US_MMS, "GPP", color = "green")
p_sm <- plot_1var(GPP_US_MMS, "SM", color = "red")
p_prcp <- plot_1var(GPP_US_MMS, "prcp", color = "blue")

p <- ggplot_multiaxis(p_gpp, p_sm, p_prcp, x = -0.02, linewidth = 1.2)
p
```

GOF *GOF*

Description

Good of fitting

Usage

```
GOF(obs, sim, w, include.cv = FALSE, include.r = TRUE)

KGE(obs, sim, w = c(1, 1, 1), ...)

NSE(obs, sim, w, ...)
```

Arguments

obs	Numeric vector, observations
sim	Numeric vector, corresponding simulated values
w	Numeric vector, weights of every points. If w included, when calculating mean, Bias, MAE, RMSE and NSE, w will be taken into considered.
include.cv	If true, cv will be included.
include.r	If true, r and R2 will be included.
...	ignored

Value

- RMSE root mean square error
- NSE NASH coefficient
- MAE mean absolute error
- AI Agreement index (only good points (w == 1)) participate to calculate. See details in Zhang et al., (2015).
- Bias bias
- Bias_perc bias percentage
- n_sim number of valid obs
- cv Coefficient of variation
- R2 correlation of determination
- R pearson correlation
- pvalue pvalue of R

References

1. https://en.wikipedia.org/wiki/Coefficient_of_determination
2. https://en.wikipedia.org/wiki/Explained_sum_of_squares
3. https://en.wikipedia.org/wiki/Nash%E2%80%93Sutcliffe_model_efficiency_coefficient
4. Zhang Xiaoyang (2015), http://dx.doi.org/10.1016/j.rse.2014.10.012

Examples

```
obs = rnorm(100)
sim = obs + rnorm(100)/4
GOF(obs, sim)
```

---

guide_coloursteps2 *guide_coloursteps2 #' @param title A title for the guide.*

Description

guide_coloursteps2 #' @param title A title for the guide.

Usage

```
guide_coloursteps2(
  title = waiver(),
  theme = NULL,
  alpha = NA,
  even.steps = TRUE,
  show.limits = NULL,
  direction = NULL,
  reverse = FALSE,
  order = 0,
  available_aes = c("colour", "color", "fill"),
  barheight = unit(0.9, "npc"),
  ...
)

guide_colorsteps2(
  title = waiver(),
  theme = NULL,
  alpha = NA,
  even.steps = TRUE,
  show.limits = NULL,
  direction = NULL,
  reverse = FALSE,
  order = 0,
  available_aes = c("colour", "color", "fill"),
```

```
  barheight = unit(0.9, "npc"),
  ...
)
```

Arguments

theme	A theme object for rendering the guide.
alpha	Alpha transparency level.
even.steps	Logical; should the steps be evenly spaced?
show.limits	Logical; should the limits be shown?
direction	Direction of the guide (”horizontal” or ”vertical”).
reverse	Logical; should the guide be reversed?
order	Order of the guide.
available_aes	Available aesthetics for this guide.
barheight	Height of the color bar. Deprecated in ggplot2 3.5.0.
...	Additional arguments passed to the guide. #’ @example R/examples/ex-stat_cut.R

---

layer_barchart *layer_barchart*

Description

layer_barchart

Usage

```
layer_barchart(
  mapping = NULL,
  data = NULL,
  brks,
  cols,
  x = 0,
  y = 0,
  width = unit(0.5, "npc"),
  height = unit(0.5, "npc"),
  just = c(0, 0),
  fontsize = 12,
  theme = NULL,
  ...
)

geom_barchart(
  mapping = NULL,
  data = NULL,
```

```
  brks,
  cols,
  x = 0,
  y = 0,
  width = unit(0.5, "npc"),
  height = unit(0.5, "npc"),
  just = c(0, 0),
  fontsize = 12,
  theme = NULL,
  ...
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `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)`).
`x`	A numeric vector or unit object specifying x-location.
`y`	A numeric vector or unit object specifying y-location.
`width`	A numeric vector or unit object specifying width.
`height`	A numeric vector or unit object specifying height.
`just`	Adjustment for column placement. Set to `0.5` by default, meaning that columns will be centered about axis breaks. Set to `0` or `1` to place columns to the left/right of axis breaks. Note that this argument may have unintended behaviour when used with alternative positions, e.g. `position_dodge()`.
`...`	Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.

- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

Examples

```
library(gg.layers)
library(ggplot2)
library(rcolors)

data("d_trendPerc")
d_mask <- mutate(d_trendPerc, mask = perc <= 0.99) # %>% as.data.frame()

brks = seq(0.9, 1, 0.025)
nbrk = length(brks) - 1
cols <- get_color(rcolors$amwg256, nbrk)

# the part of not significant
ggplot(data = d_mask, aes(x, y)) +
  geom_raster(aes(fill = perc)) +
  layer_barchart(aes(z = perc),
    width = unit(0.3, "npc"),
    height = unit(0.3, "npc"),
    brks = brks, cols = cols)
```

`layer_PosNeg` *layer_PosNeg*

Description

layer_PosNeg

Usage

```
layer_PosNeg(mapping = NULL, data = NULL, x = 0, y = 0, size = 12, ...)

add_PosNeg(
  z,
  x = 0.5,
```

```
  y = 0.5,
  height.factor = 1.2,
  cols = c("blue", "red"),
  alpha = 0.8,
  ...
)

layer_PosNeg_sign(
  mapping = NULL,
  data = NULL,
  x = 0.5,
  y = 0.5,
  height.factor = 1.2,
  size = 12,
  ...
)

add_PosNeg_sign(
  z,
  mask,
  x = 0.5,
  y = 0.5,
  height.factor = 1.2,
  cols = c("blue", "red"),
  alpha = 0.8,
  ...
)
```

Arguments

`...`	other parameters to `element_grob_text`

`make_colorbar` *make_colorbar*

Description

make_colorbar

Usage

```
make_colorbar(
  at,
  labels = NULL,
  labeller = format,
  space = "right",
  width = 2,
```

```
  height = 1,
  col = NULL,
  alpha = 1,
  pretty = FALSE,
  equispaced = TRUE,
  tick.number = 7,
  tck = 0.3,
  tck.padding = 0,
  raster = FALSE,
  interpolate = FALSE,
  tri.upper = NA,
  tri.lower = NA,
  legend.line = element_line(linewidth = 0.8),
  legend.box = element_rect(linewidth = 0.5),
  hjust = 0.5,
  vjust = 0.5,
  size = 12,
  family = "Times",
  legend.text.location = c(0.5, 0.5),
  legend.margin = bb_margin(),
  title = NULL,
  legend.text = element_text(hjust = 0.5),
  legend.title = element_text(),
  fct.title.height = 1.8,
  padding.left = unit(2, "points"),
  padding.right = unit(2, "points"),
  ...,
  draw = FALSE,
  vp = NULL
)
```

Arguments

`legend.box`	arrangement of multiple legends ("horizontal" or "vertical")
`hjust`, `vjust`	used in `grid::grid.layout()`
`legend.margin`	the margin around each legend (`margin()`); inherits from `margins`.
`title`	all title elements: plot, axes, legends (`element_text()`; inherits from `text`)
`legend.text`	• `cex`: • `col`: • `font`: • `fontfamily`: The font family • `fontface`: The font face (bold, italic, ...) • `lineheight`:
`legend.title`	title of legend (`element_text()`; inherits from `title`)
`padding.left`, `padding.right`	padding in the left and right of the legend

...	additional element specifications not part of base ggplot2. In general, these should also be defined in the `element tree` argument. Splicing a list is also supported.
`draw`	A logical value indicating whether graphics output should be produced.
`vp`	A Grid viewport object (or NULL).

Examples

```
library(ggplot2)
library(rcolors)
library(magrittr)
library(glue)

## example 01
spaces <- c("right", "left", "top", "bottom") # %>% set_names(., .)

make_cbar <- function(brks, space, outfile = NULL) {
  # brks <- 1:5
  nbrk <- length(brks) - 1
  cols <- rcolors::get_color("amwg256", nbrk)

  g <- make_colorbar(
    at = brks, col = cols,
    space = space, title = space
  )
  if (!is.null(outfile)) {
    if (require(Ipaper))
      Ipaper::write_fig(g, outfile, 10, 6)
  }
  g
}

brks = 1:5 %>% c(-Inf, ., Inf)
spaces <- c("right", "left", "top", "bottom")
ps = lapply(spaces, function(space) {
  fout = glue("cbar_{space}.pdf")
  fout = NULL
  make_cbar(brks, space, fout)
})
g = cowplot::plot_grid(plotlist = ps)
# g = patchwork::wrap_plots(ps)
# grid.newpage(); grid.draw(g)
# Ipaper::write_fig(g, "Rplot.pdf", 10, 6) # uncomment if want to show figure


## example 02
brks = c(-Inf, -1, 0, 1, 3, 6, 9, Inf)
# brks = 1:10
nbrk = length(brks) - 1
cols = get_color(rcolors$amwg256, nbrk)

spaces = c("right", "left", "top", "bottom") #%>% set_names(., .)
```

```
params <- list(
  at = brks, col = cols, height = 1,
  tck = 0.4,
  # padding.left = unit(2, "points"),
  # padding.right = unit(2, "points"),
  space = spaces[1],
  # legend.line = element_line(size = 0.1, linetype = 1, color = "black"),
  # legend.text = element_text(hjust = 0.5),
  legend.text.location = c(0.2, 0.5),
  # legend.margin = bb_margin(t = 0.1),
  # legend.text.just = c(0.5, 0.5),
  # title = NULL,
  title = "d/decade",
  fct.title.height = 3,
  legend.title = element_text(size = 14)
  # legend.box = element_rect(size = 0.5),
  # legend.line = element_line(size = 1),
  # legend.text = list(fontfamily = "Times", cex = 1.1),
  # hjust = 0.5
)
cbar <- do.call(make_colorbar, params)
# write_fig(cbar, "a.pdf", 0.9, 6)

# cowplot::plot_grid(plotlist = lst)
p <- ggplot(mtcars, aes(mpg, disp)) + geom_point() +
  facet_wrap(~cyl)
p + cbar

p <- ggplot(mtcars, aes(mpg, disp)) + geom_point() +
  facet_wrap(~cyl, nrow = 2)
add_colorbar(p, cbar)

## Test the bottom
params$space = "bottom"
params$title = ""
cbar2 <- do.call(make_colorbar, params)
add_colorbar(p, cbar2, space = "bottom",
             title = "(mm/y)",
             legend.title = element_text(hjust = -5, vjust = -3, family = "Times"))

# Another option
title = element_grob_text(element_text(family = "Times", hjust = 1, vjust = 0, size = 12),
  label = "(mm/y)", x = 0.98, y = 0.09)
add_colorbar(p, cbar2, space = "bottom") %>% add_grob(title)
```

---

scale_fill_gradientn2

scale_fill_gradientn2

Description

scale_fill_gradientn2

Usage

```
scale_fill_gradientn2(
  ...,
  colours,
  values = NULL,
  space = "Lab",
  na.value = "transparent",
  guide = colourbar_triangle(),
  oob = censor2,
  aesthetics = "colour",
  colors
)
```

Arguments

`...` Arguments passed on to `continuous_scale`

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

`breaks` One of:

- `NULL` for no breaks
- `waiver()` for the default breaks computed by the transformation object
- A numeric vector of positions
- A function that takes the limits as input and returns breaks as output (e.g., a function returned by `scales::extended_breaks()`). Note that for position scales, limits are provided after scale expansion. Also accepts rlang lambda function notation.

`minor_breaks` One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (none for discrete, one minor break between each major break for continuous)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation. When the function has two arguments, it will be given the limits and major break positions.

`n.breaks` An integer guiding the number of major breaks. The algorithm may choose a slightly different number to ensure nice break labels. Will only have an effect if `breaks = waiver()`. Use `NULL` to use the default number of breaks given by the transformation.

`labels` One of the options below. Please note that when `labels` is a vector, it is highly recommended to also set the `breaks` argument as a vector to protect against unintended mismatches.

- 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)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

limits One of:

- NULL to use the default scale range
- A numeric vector of length two providing limits of the scale. Use NA to refer to the existing minimum or maximum
- A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang lambda function notation. Note that setting limits on positional scales will **remove** data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see coord_cartesian()).

rescaler A function used to scale the input values to the range [0, 1]. This is always scales::rescale(), except for diverging and n colour gradients (i.e., scale_colour_gradient2(), scale_colour_gradientn()). The rescaler is ignored by position scales, which always use scales::rescale(). Also accepts rlang lambda function notation.

oob One of:

- Function that handles limits outside of the scale limits (out of bounds). Also accepts rlang lambda function notation.
- The default (scales::censor()) replaces out of bounds values with NA.
- scales::squish() for squishing out of bounds values into range.
- scales::squish_infinite() for squishing infinite values into range.

trans [**Deprecated**] Deprecated in favour of transform.

call The call used to construct the scale for reporting messages.

super The super class to use for the constructed scale

colours, colors

Vector of colours to use for n-colour gradient.

values if colours should not be evenly positioned along the gradient this vector gives the position (between 0 and 1) for each colour in the colours vector. See rescale() for a convenience function to map an arbitrary range to between 0 and 1.

space colour space in which to calculate gradient. Must be "Lab" - other values are deprecated.

na.value Colour to use for missing values

guide Type of legend. Use "colourbar" for continuous colour bar, or "legend" for discrete colour legend.

`oob`	function that handles limits outside of the scale limits
`aesthetics`	Character string or vector of character strings listing the name(s) of the aesthetic(s) that this scale works with. This can be useful, for example, to apply colour settings to the `colour` and `fill` aesthetics at the same time, via `aesthetics = c("colour", "fill")`.

`st_hatched_polygon` *st_hatched_polygon*

Description

st_hatched_polygon

Usage

```
st_hatched_polygon(x, density = 2, angle = 45, fillOddEven = FALSE)
```

Arguments

`x`	sf polygon object
`density`	the density of shading lines, in lines per inch. The default value of NULL means that no shading lines are drawn. A zero value of density means no shading nor filling whereas negative values and NA suppress shading (and so allow color filling).
`angle`	the slope of shading lines, given as an angle in degrees (counter-clockwise).
`fillOddEven`	logical controlling the polygon shading mode: see below for details. Default `FALSE`.

Value

An hatched area, sf lines

References

1. https://github.com/statnmap/HatchedPolygons
2. https://statnmap.github.io/HatchedPolygons/
3. https://statnmap.github.io/HatchedPolygons/articles/leaflet_shading_polygon.html

Examples

```
library(gg.layers)
library(ggplot2)
data("d_trendPerc")

d = d_trendPerc %>% subset(perc >= 0.99) %>% .[, 1:2]
poly = st_point2poly(d)
hatches = st_hatched_polygon(poly) #
ggplot(hatches) + geom_sf()
```

st_point2poly *st_point2poly*

Description

st_point2poly

Usage

```
st_point2poly(xyz, crs = 4326)

df2rast(xyz)

rast2poly(r, crs = 4326)

st_dissolve(x, by = NULL, ...)
```

Arguments

xyz	matrix or data.frame with at least three columns: x and y coordinates, and values (z). There may be several 'z' variables (columns)
crs	one of (i) character: a string accepted by GDAL, (ii) integer, a valid EPSG value (numeric), or (iii) an object of class crs.

References

1. https://gis.stackexchange.com/questions/192771/how-to-speed-up-raster-to-polygon-conversion-in-r

See Also

raster::rasterFromXYZ()

stat_cut *stat_cut*

Description

stat_cut

Usage

```
stat_cut(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  breaks = NULL,
  include.lowest = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

StatCut
```

Format

An object of class `StatCut` (inherits from `StatIdentity`, `Stat`, `ggproto`, `gg`) of length 6.

Examples

```
library(rcolors)
library(ggplot2)
library(magrittr)

df = data.frame(x = 1:10, y = 1:10, z = 1:10)

brks <- c(2, 4, 6, 8) %>% c(-Inf, ., Inf) #
nbrk <- length(brks) - 1
cols = get_color(rcolors$amwg256, nbrk)

ggplot(df, aes(x, y, z = x)) +
  stat_cut(aes(color = after_stat(level)), breaks = brks, geom = "point") +
  scale_color_manual(
    values = cols,
    guide = guide_coloursteps2(title = "lgd", barheight = unit(0.8, "npc"))
  )

## another option: use `scale_color_stepsn`
# example 2
# ! unable to accurately control the used colors
ggplot(df, aes(x, y, color = x)) +
  geom_point() +
  scale_color_stepsn(
    colors = cols,
    breaks = brks,
    guide = guide_coloursteps2(title = "lgd")
  ) +
  theme(
    legend.title = element_blank(),
```

```
    legend.margin = margin(l = -2)
  )

# example 3
df <- expand.grid(X1 = 1:10, X2 = 1:10)
df$value <- df$X1 * df$X2

brks = c(10, 15, 25, 50)
nbrk <- length(brks) + 1
cols = get_color(rcolors$amwg256, nbrk)

# This can be changed with the `even.steps` argument
ggplot(df, aes(X1, X2)) +
  geom_tile(aes(fill = value)) +
  scale_fill_stepsn(
    colors = cols, breaks = brks,
    guide = guide_colorsteps2()
  ) +
  theme(
    legend.title = element_blank(),
    legend.margin = margin(l = -2)
  )
```

---

stat_gof *stat_gof*

Description

stat_gof

Usage

```
stat_gof(
  mapping = NULL,
  data = NULL,
  geom = GeomRichTextNpc,
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  show.bias = TRUE,
  label.format = fmt_gof,
  x = 0.05,
  y = 0.95,
  inherit.aes = TRUE,
  ...
)

geom_gof(
  mapping = NULL,
```

```
  data = NULL,
  stat = StatGOF,
  position = "identity",
  ...,
  show.bias = TRUE,
  label.format = fmt_gof,
  x = 0,
  y = 1,
  hjust = 0,
  vjust = 1,
  size = 5,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_reg(
  mapping = NULL,
  data = NULL,
  formula = y ~ x,
  digits = 2,
  units = "",
  format = paste0("Slope = {str_num(slope, digits)}{unit}",
    ", p-value = {str_num(pvalue, digits)}"),
  fun_slope = NULL,
  x = 0,
  y = 1,
  hjust = 0,
  vjust = 1,
  mar = 0.03,
  height.factor = 1.2,
  family = "Times",
  color = NULL,
  position = "dodge",
  ...
)
```

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)`).
`position`	"dodge" or "identity"
`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.
`show.bias`	whether to show bias
`label.format`	format string for label, default `fmt_gof`
`x`	A numeric vector or unit object specifying x-values.
`y`	A numeric vector or unit object specifying y-values.
`inherit.aes`	If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.
`...`	Other arguments passed on to `layer()`. 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.
`stat`	The statistical transformation to use on the data for this layer, as a string.
`hjust`	Horizontal justification (in $[0, 1]$)
`vjust`	Vertical justification (in $[0, 1]$)
`formula`	an object of class `"formula"` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
`digits`	the number of *significant* digits to be passed to `format(coef(x), .)` when `print()`ing.
`units`	A Vector with the length of subplots
`family`	The typeface to use. The validity of this value will depend on the graphics device being used for rendering the plot. See the systemfonts vignette for guidance on the best way to access fonts installed on your computer. The values `"sans"`, `"serif"`, and `"mono"` should always be valid and will select the default typeface for the respective styles. However, what is considered default is dependant on the graphics device and the operating system.

Details

- `b`: the object returned by `broom::tidy()`
- `s`: the object returned by `broom::glance()`
  - `*R*^2 = {str_num(s$r.squared, digits)}`

- slope:
- pvalue:
- pcode: significant code, e.g., **, *, -

Value

No return. This function is used to calculate data for gglot2 geom_*, just like ggplot2::stat_smooth().

Examples

```
library(ggplot2)
library(data.table)

dat <- data.table(mtcars)
dat$cyl <- as.factor(dat$cyl)
table(dat$cyl)

ggplot(dat, aes(wt, mpg, color = cyl)) +
    geom_point() +
    stat_reg(data = dat[cyl != 4], y = 1, mar = 0, position = "none") +
    facet_wrap(~cyl)

ggplot(dat, aes(wt, mpg, color = cyl)) +
  geom_point() +
  stat_reg(data = dat[cyl != 4], y = 1,
               position = "dodge",
               height.factor = 1.2,
               unit = "gC m^-2 d^-1")

ggplot(dat, aes(wt, mpg, color = cyl)) +
  geom_point() +
  stat_reg(data = dat[cyl != 4], y = 1,
               position = "none",
               height.factor = 1.2,
               unit = "gC m^-2 d^-1") +
  facet_wrap(~cyl)


ggplot(dat, aes(wt, mpg, color = cyl)) +
    geom_point() +
    stat_gof(x = 0, y = 1) +
    # stat_reg(data = subset(dat, cyl == 4), y = 1, color = "red") +
    # stat_reg(data = subset(dat, cyl == 6), y = 0.8) +
    facet_wrap(~cyl)
```

Description

stat_gof2

Usage

```
stat_gof2(
  mapping = NULL,
  data = NULL,
  geom = GeomRichTextNpc,
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  show.bias = TRUE,
  label.format = fmt_gof,
  x = 0.05,
  y = 0.95,
  inherit.aes = TRUE,
  ...
)

geom_gof2(
  mapping = NULL,
  data = NULL,
  stat = StatGOF2,
  position = "identity",
  ...,
  show.bias = TRUE,
  label.format = fmt_gof,
  eval.flood = FALSE,
  x = 0,
  y = 1,
  hjust = 0,
  vjust = 1,
  size = 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)`).
`position`	Position adjustment, either as a string, or the result of a call to a position adjustment function. Cannot be jointy specified with `nudge_x` or `nudge_y`.
`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.
`label.format`	default value: `"*NSE* = {str_num(NSE,2)}, *R^2* = {str_num(R2, 2)} \n *RMSE* = {str_num(RMSE,2)}"`. `label.format` will be evaluated by `glue::glue()`. Variables inside `{}` will be evaluated. All variables returned by `GOF()` are supported.
`x`	A numeric vector or unit object specifying x-values.
`y`	A numeric vector or unit object specifying y-values.
`inherit.aes`	If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.
`...`	Other arguments passed on to `layer()`. 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.
`stat`	The statistical transformation to use on the data for this layer, as a string.

Value

No return. This function is used to calculate data for gglot2 `geom_*`, just like `ggplot2::stat_smooth()`.

required axes

- `obs`: observed
- `sim`: simulated

Examples

```
library(ggplot2)
library(data.table)

dates <- seq(as.Date("2010-01-01"), length.out = 32, by = "day")
dat <- data.table(mtcars) %>% cbind(date = dates, .)
```

```
dat$cyl <- as.factor(dat$cyl)
# table(dat$cyl)

ggplot(dat, aes(date, mpg, color = cyl)) +
  stat_gof2(aes(obs = mpg, sim = wt), x = 0, y = 1) +
  geom_point() +
  facet_wrap(~cyl)

## Example 2
dat = GPP_US_MMS
ggplot(dat, aes(date, GPP)) +
  geom_line() +
  stat_gof2(aes(obs=GPP, sim=SM), x = 0, y = 1)

##
dat = data.table(
  obs = c(10, 12, 15, 14, 13, 16, 18, 20, 19, 22),
  sim = c(11, 13, 14, 15, 12, 17, 17, 21, 20, 20),
  sim2 = c(11, 13, 14, 15, 12, 17, 17, 21, 20, 25),
  group = rep(c("A", "B"), each = 5)
)

showtext::showtext_auto()
fmt_gof <- "*NSE* = {str_num(NSE,2)}, *R^2* = {str_num(R2, 2)}"

p = ggplot(dat, aes(obs, sim)) +
  geom_point() +
  stat_gof2(aes(obs = obs, sim = sim), x = 0, y = 1,
    eval.flood = TRUE, show.bias = FALSE, label.format = fmt_gof, size = 6) +
  stat_gof2(aes(obs = obs, sim = sim2), x = 0, y = 0.9,
    eval.flood = TRUE, show.bias = FALSE, label.format = fmt_gof, size = 6)

# write_fig(p, "d:/Rplot.pdf", 10, 5)
```

`stat_interval` *geom_interval*

Description

geom_interval

Usage

```
stat_interval(
  mapping = NULL,
  data = NULL,
  geom = GeomInterval,
  position = "identity",
  interval = 0.8,
  fun_middle = "median",
```

```
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)

geom_interval(
  mapping = NULL,
  data = NULL,
  stat = "interval",
  position = "identity",
  ...,
  fun_middle = "median",
  interval = 0.8,
  alpha.line = 0.7,
  param_ensemble = list(),
  param_slope = list(alpha = 1, linewidth = 1),
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), 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 to display the data for this layer. When using a `stat_*()` function to construct a layer, the `geom` argument can be used to override the default coupling between stats and geoms. The `geom` argument accepts the following: • A `Geom` ggproto subclass, for example `GeomPoint`. • A string naming the geom. To give the geom as a string, strip the function name of the `geom_` prefix. For example, to use `geom_point()`, give the geom as `"point"`. • For more information and other ways to specify the geom, see the layer geom documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`interval` interval of confidence interval, default is 0.8

`fun_middle` function to calculate middle value, default is median

`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. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`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. `annotation_borders()`.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.

- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`stat` The statistical transformation to use on the data for this layer. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following:

- A `Stat` ggproto subclass, for example `StatCount`.
- A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`.
- For more information and other ways to specify the stat, see the layer stat documentation.

`alpha.line` alpha of ensemble line

`param_ensemble` list of parameters for ensemble line

`param_slope` list of parameters for slope line

Examples

```
library(ggplot2)
df <- CMIP6_HWD

p <- ggplot(df, aes(year, value, color = prob, fill = prob))

# both line and fill
p + geom_interval(alpha = 0.2, alpha.line = 1,
  linewidth = 0.7, linetype = 2, # for ensemble line
  param_slope = list(linewidth = 1, alpha = 1, linetype = 1), # slope line
  interval = 0.8, fun_middle = "mean")

# only line
p + geom_interval(
  fill = "NA",
  alpha = 0.4, interval = 0.8, fun_middle = "mean", linewidth = 0.4
) +
  theme(legend.key = element_rect(fill = "NA"))

# only fill
p + geom_interval(
  color = "NA", key_glyph = "polygon2",
  alpha = 0.4, interval = 0.8, fun_middle = "mean", linewidth = 0.4
) +
  theme(legend.key = element_rect(fill = "NA"))
```

`stat_prob_2d` *prob density*

Description

prob density

Usage

```
stat_prob_2d(
  mapping = NULL,
  data = NULL,
  geom = "density_2d",
  position = "identity",
  ...,
  contour = TRUE,
  contour_var = "density",
  n = 100,
  h = NULL,
  adjust = c(1, 1),
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), 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 stat_prob_2d.
`position`	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following: • The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.

- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Arguments passed on to `geom_contour`

`binwidth` The width of the contour bins. Overridden by `bins`.

`bins` Number of contour bins. Overridden by `breaks`.

`breaks` One of:

- Numeric vector to set the contour breaks
- A function that takes the range of the data and binwidth as input and returns breaks as output. A function can be created from a formula (e.g. ~ fullseq(.x, .y)).

Overrides `binwidth` and `bins`. By default, this is a vector of length ten with `pretty()` breaks.

`contour` If `TRUE`, contour the results of the 2d density estimation.

`contour_var` Character string identifying the variable to contour by. Can be one of `"density"`, `"ndensity"`, or `"count"`. See the section on computed variables for details.

`n` Number of grid points in each direction.

`h` Bandwidth (vector of length two). If `NULL`, estimated using `MASS::bandwidth.nrd()`.

`adjust` A multiplicative bandwidth adjustment to be used if 'h' is 'NULL'. This makes it possible to adjust the bandwidth while still using the a bandwidth estimator. For example, `adjust = 1/2` means use half of the default bandwidth.

`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. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`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. `annotation_borders()`.

Examples

```
library(ggplot2)

# Data
n = 200
set.seed(1)
```

```
a <- data.frame( x=rnorm(n, 10, 1.9), y=rnorm(n, 10, 1.2) )
b <- data.frame( x=rnorm(n, 14.5, 1.9), y=rnorm(n, 14.5, 1.9) )
c <- data.frame( x=rnorm(n, 9.5, 1.9), y=rnorm(n, 15.5, 1.9) )
data <- rbind(a,b,c)

# Show the area only
ggplot(data, aes(x=x, y=y) ) +
  stat_prob_2d(aes(fill = ..level.., color =..level.. ), geom = "polygon")

ggplot(data, aes(x=x, y=y) ) +
  stat_prob_2d(aes(fill = ..level..), geom = "polygon", color = "white")

ggplot(data, aes(x=x, y=y) ) +
  stat_prob_2d(aes(color =..level.. ), geom = "path")
```

---

stat_signHatch *geom_signHatch*

Description

geom_signHatch

Usage

```
stat_signHatch(
  mapping = NULL,
  data = NULL,
  geom = "sf",
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...,
  density = 1,
  angle = 45
)

geom_signHatch(
  mapping = aes(),
  data = NULL,
  stat = "signHatch",
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...,
  density = 1,
  angle = 45
)
```

Arguments

mapping	Set of aesthetic mappings created by 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 to display the data for this layer. When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following: • A Geom ggproto subclass, for example GeomPoint. • A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use geom_point(), give the geom as "point". • For more information and other ways to specify the geom, see the layer geom documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
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. You can also set this to one of "polygon", "line", and "point" to override the default legend.
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. annotation_borders().
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further

arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`density` the density of shading lines, in lines per inch. The default value of NULL means that no shading lines are drawn. A zero value of density means no shading nor filling whereas negative values and NA suppress shading (and so allow color filling).

`angle` the slope of shading lines, given as an angle in degrees (counter-clockwise).

`stat` The statistical transformation to use on the data for this layer. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following:

- A `Stat` ggproto subclass, for example `StatCount`.
- A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`.
- For more information and other ways to specify the stat, see the layer stat documentation.

Aesthetics

`geom_signHatch()` requires the following aesthetics:

- `x`:
- `y`:
- `mask`:

Examples

```
library(gg.layers)
library(ggplot2)
data("d_trendPerc")

d_mask <- mutate(d_trendPerc, mask = perc <= 0.99) #%>% as.data.frame()

ggplot() +
    geom_raster(data = d_trendPerc, aes(x, y, fill = perc)) +
    # geom_sf(data = shp) +
    geom_signHatch(data = d_mask, aes(x, y, mask = mask), color = "red")
```

---

stat_signPattern *stat_signPattern*

Description

stat_signPattern

Usage

```
stat_signPattern(
  mapping = NULL,
  data = NULL,
  geom = "sf_pattern",
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `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)`).

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`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. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`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. `annotation_borders()`.

`...` other parameters to `ggpattern::geom_sf_pattern()`

See Also

`ggpattern::geom_sf_pattern()`

Examples

```
library(gg.layers)
library(ggplot2)
data("d_trendPerc")

d_mask <- mutate(d_trendPerc, mask = perc <= 0.99) # %>% as.data.frame()

# significant part; geom_sf_pattern still has bug unsolved.
ggplot() +
  geom_raster(data = d_trendPerc, aes(x, y, fill = perc)) +
  stat_signPattern(data = d_mask, aes(x, y, mask = mask),
                   fill = "transparent", color = "red",
                   pattern_density = 0.02)

# insignificant
ggplot() +
    geom_raster(data = d_trendPerc, aes(x, y, fill = perc)) +
    stat_signPattern(data = d_mask, aes(x, y, mask = !mask),
                     fill = "transparent", color = "red",
                     pattern_density = 0.02) #-> p
# Ipaper::write_fig(p, "temp.pdf")
```

stat_signPoint *stat_signPoint*

Description

stat_signPoint

Usage

```
stat_signPoint(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...,
  fact = 1
)

geom_signPoint(
  mapping = aes(),
  data = NULL,
  stat = "signPoint",
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...,
  fact = 1
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `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 to display the data for this layer. When using a `stat_*()` function to construct a layer, the `geom` argument can be used to override the default coupling between stats and geoms. The `geom` argument accepts the following:

- A `Geom` ggproto subclass, for example `GeomPoint`.
- A string naming the geom. To give the geom as a string, strip the function name of the `geom_` prefix. For example, to use `geom_point()`, give the geom as `"point"`.
- For more information and other ways to specify the geom, see the layer geom documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`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.

You can also set this to one of "polygon", "line", and "point" to override the default legend.

`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. `annotation_borders()`.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.

- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`fact` postive integer. Plot one point in `fact` steps.

`stat` The statistical transformation to use on the data for this layer. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following:

- A `Stat` ggproto subclass, for example `StatCount`.
- A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`.
- For more information and other ways to specify the stat, see the layer stat documentation.

Aesthetics

`geom_signPoint()` requires the following aesthetics:

- `x`:
- `y`:
- `mask` (optional): Boolean, `mask=TRUE` is regarded as significant. Only plot regions with `mask = TRUE`.

Examples

```
library(gg.layers)
library(ggplot2)
data("d_trendPerc")

d_mask <- mutate(d_trendPerc, mask = perc <= 0.99) # %>% as.data.frame()

# the part of not significant
ggplot(data = d_mask, aes(x, y)) +
  geom_raster(aes(fill = perc)) +
  geom_signPoint(aes(mask = !mask), fact = 2, shape = 4)

# significant
ggplot(data = d_mask, aes(x, y)) +
  geom_raster(aes(fill = perc)) +
  geom_signPoint(aes(mask = mask), fact = 2)

## Another option
fact = 2
```

```
ggplot(data = d_mask, aes(x, y)) +
  geom_raster(aes(fill = perc)) +
  geom_point(data = ~ resample_points(.x, fact))
```

`str_mk` *markdown superscript and subscript*

Description

markdown superscript and subscript

Usage

```
str_mk(x)

label_mk(labels, ...)
```

Arguments

`x`	character vector
`labels`	character vector
`...`	ignored

Author(s)

Dongdong Kong

Examples

```
x <- c(
  "gC m^{-2} d^{-1}",
  "gC m^-2 d^-1",
  "gC m_{-2} d_{-1}",
  "gC m_-2 d_-1",
  "gC \n mm/d"
)
str_mk(x)
## use str_mk in ggplot
library(ggplot2)

d <- data.frame(
  x = 1:3, y = 1:3,
  varname = c("T_min", "T_max", "T_avg")
)

ggplot(d, aes(x, y)) +
  geom_point() +
  geom_richtext(aes(label = varname), x = 2, y = 2) +
  facet_wrap(~varname, labeller = label_mk) +
```

```
  theme(
    strip.text = element_markdown(face = "bold")
  )
```

`str_num` *Rounding of Numbers, and convert to string*

Description

Rounding of Numbers, and convert to string

Usage

```
str_num(x, digits = 2)
```

Arguments

`x`	a numeric vector. Or, for `round` and `signif`, a complex vector.
`digits`	integer indicating the number of decimal places (`round`) or significant digits (`signif`) to be used. For `round`, negative values are allowed (see 'Details').

Examples

```
format(round(2, 2)) # 2
str_num(2, 2) # 2.00
```

Index