Package 'ggmulti'

October 8, 2025

Type Package

Title High Dimensional Data Visualization

Version 1.0.8

Description It provides materials (i.e. 'serial axes' objects, Andrew's plot, various glyphs for scatter plot) to visualize high dimensional data.

License GPL-2

Depends R (>= 3.4.0), methods, ggplot2 (>= 3.5.2)

Imports stats, utils, grid, dplyr, tidyr, cli

Suggests png, tools, stringr, markdown, magrittr, gridExtra, tibble, testthat, grDevices, knitr, rmarkdown, tidyverse, gtable, covr, maps, nycflights13, ggplot2movies

RoxygenNote 7.3.2

LazyData true

Encoding UTF-8

VignetteBuilder knitr

Language en-US

NeedsCompilation no

Author Zehao Xu [aut, cre], R. Wayne Oldford [aut], Teun van den Brand [ctb]

Maintainer Zehao Xu <z267xu@gmail.com>

Repository CRAN

Date/Publication 2025-10-08 08:40:07 UTC

Contents

`add_serialaxes_layers` *Layers for serial axes coordinate*

Description

Project the regular `geom` layers onto the serial axes coordinate.

Usage

```
add_serialaxes_layers(layer, plot, object, axes)
```

Arguments

`layer`	a layer object
`plot`	a `ggplot` object
`object`	some parameters used to modify this serial axes `ggplot` object (i.e. `axes.sequence`, ...)
`axes`	canvas sequence axes

Details

The class is determined by layers you add. For example, you want to add a boxplot layer on serial axes coordinate. By the ggplot syntax, it should be `ggplot(data, mapping) + geom_boxplot() + coord_serialaxes()` To make it work, object `add_serialaxes_layers.GeomBoxplot` must be created. In this function, some computations will be applied.

coord_radial *Radial axes*

Description

A radial (spider) coordinate. A wrapper of the function `coord_polar()` by forcing it linear.

Usage

```
coord_radial(theta = "x", start = 0, direction = 1, clip = "on")
```

Arguments

`theta`	variable to map angle to (`x` or `y`)
`start`	Offset of starting point from 12 o'clock in radians. Offset is applied clockwise or anticlockwise depending on value of `direction`.
`direction`	1, clockwise; -1, anticlockwise
`clip`	Should drawing be clipped to the extent of the plot panel? A setting of `"on"` (the default) means yes, and a setting of `"off"` means no. For details, please see `coord_cartesian()`.

Details

The serial histogram and serial density cannot be applied on a radial coordinate yet.

Examples

```
if(require("dplyr")) {
ggplot(NBAstats2021, mapping = aes(colour = Playoff)) +
  geom_serialaxes(
    axes.sequence = c("PTS", "OPTS", "3PM", "O3PM", "PTS"),
      scaling = "variable"
    ) +
  coord_radial() +
  scale_x_continuous(
    breaks = 1:5,
    labels = c("Points",
               "Oppo Points",
               "3P Made",
               "Oppo 3P Made",
               "Points Per Game")) +
  scale_y_continuous(labels = NULL) +
  facet_wrap(~CONF)
  }
```

coord_serialaxes *Serial axes coordinates*

Description

It is mainly used to visualize the high dimensional data set either on the parallel coordinate or the radial coordinate.

Usage

```
coord_serialaxes(
  axes.layout = c("parallel", "radial"),
  scaling = c("data", "variable", "observation", "none"),
  axes.sequence = character(0L),
  positive = TRUE,
  ...
)
```

Arguments

axes.layout	Serial axes layout, either "parallel" or "radial".
scaling	One of data, variable, observation or none (not suggested the layout is the same with data) to specify how the data is scaled.
axes.sequence	A vector with variable names that defines the axes sequence.
positive	If y is set as the density estimate, where the smoothed curved is faced to, right (positive) or left (negative) as vertical layout; up (positive) or down (negative) as horizontal layout?
...	other arguments used to modify layers

Details

Serial axes coordinate system (parallel or radial) is different from the Cartesian coordinate system or its transformed system (say polar in ggplot2) since it does not have a formal transformation (i.e. in polar coordinate system, "x = rcos(theta)", "y = rsin(theta)"). In serial axes coordinate system, mapping aesthetics does not really require "x" or "y". Any "non-aesthetics" components passed in the mapping system will be treated as an individual axis.

To project a common geom layer on such serialaxes, users can customize function add_serialaxes_layers.

Value

a ggproto object

Potential Risk

In package ggmulti, the function ggplot_build.gg is provided. At the ggplot construction time, the system will call ggplot_build.gg first. If the plot input is not a CoordSerialaxes coordinate system, the next method ggplot_build.ggplot will be called to build a "gg" plot; else some geometric transformations will be applied first, then the next method ggplot_build.ggplot will be executed. So, the potential risk is, if some other packages e.g. foo, also provide a function ggplot_build.gg that is used for their specifications but the namespace is beyond the ggmulti (ggmulti:::ggplot_build.gg is covered), error may occur. If so, please consider using the geom_serialaxes.

Examples

```
if(require("dplyr")) {
# Data
nba <- NBAstats2021 %>%
  mutate(
    dPTS = PTS - OPTS,
    dREB = REB - OREB,
    dAST = AST - OAST,
    dTO = TO - OTO
  )
# set sequence by `axes.sequence`
p <- ggplot(nba,
            mapping = aes(
              dPTS = dPTS,
              dREB = dREB,
              dAST = dAST,
              dTO = dTO,
              colour = Win
            )) +
       geom_path(alpha = 0.2) +
       coord_serialaxes(axes.layout = "radial") +
       scale_color_gradient(low="blue", high="red")
p
# quantile layer
p + geom_quantiles(quantiles = c(0.5),
                   colour = "green", linewidth = 1.2)

# facet
p +
  facet_grid(Playoff ~ CONF)
}
```

---

dot_product *Transformation Coefficients*

Description

The dimension of the original data set is n*p. It can be projected onto a n*k space. The functions below are to provide such transformations, e.g. the Andrews coefficient (a Fourier transformation) and the Legendre polynomials.

Usage

```
andrews(p = 4, k = 50 * (p - 1), ...)

legendre(p = 4, k = 50 * (p - 1), ...)
```

Arguments

p	The number of dimensions
k	The sequence length
...	Other arguments passed on to methods. Mainly used for customized transformation function

Value

A list contains two named components

1. vector: A length k vector (define the domain)
2. matrix: A p*k transformed coefficient matrix

References

Andrews, David F. "Plots of high-dimensional data." *Biometrics* (1972): 125-136.

Abramowitz, Milton, and Irene A. Stegun, eds. "Chapter 8" *Handbook of mathematical functions with formulas, graphs, and mathematical tables*. Vol. 55. US Government printing office, 1948.

Examples

```
x <- andrews(p = 4)
dat <- iris[, -5]
proj <- t(as.matrix(dat) %*% x$matrix)
matplot(x$vector, proj,
        type = "l", lty = 1,
        col = "black",
        xlab = "x",
        ylab = "Andrews coefficients",
        main = "Iris")
```

Geom-ggproto *Base Geom ggproto classes for ggplot2*

Description

All geom_ functions (like geom_point) return a layer that contains a Geom object (like GeomPoint). The Geom object is responsible for rendering the data in the plot. Each of the Geom objects is a ggproto object, descended from the top-level Geom, and each implements various methods and fields. Compared to Stat and Position, Geom is a little different because the execution of the setup and compute functions is split up. setup_data runs before position adjustments, and draw_layer is not run until render time, much later. This means there is no setup_params because it's hard to communicate the changes.

Usage

```
GeomDensity_

GeomBar_

GeomImageGlyph

GeomPolygonGlyph

GeomQuantiles

GeomSerialaxesDensity

GeomSerialAxesGlyph

GeomSerialaxesHist

GeomSerialaxesQuantile

GeomSerialaxes
```

Format

An object of class GeomDensity_ (inherits from GeomRibbon, Geom, ggproto, gg) of length 6.

An object of class GeomBar_ (inherits from GeomBar, GeomRect, Geom, ggproto, gg) of length 4.

An object of class GeomImageGlyph (inherits from Geom, ggproto, gg) of length 7.

An object of class GeomPolygonGlyph (inherits from Geom, ggproto, gg) of length 7.

An object of class GeomQuantiles (inherits from GeomQuantile, GeomPath, Geom, ggproto, gg) of length 1.

An object of class GeomSerialaxesDensity (inherits from GeomDensity_, GeomRibbon, Geom, ggproto, gg) of length 2.

An object of class GeomSerialAxesGlyph (inherits from Geom, ggproto, gg) of length 7.

An object of class GeomSerialaxesHist (inherits from GeomBar_, GeomBar, GeomRect, Geom, ggproto, gg) of length 2.

An object of class GeomSerialaxesQuantile (inherits from GeomPath, Geom, ggproto, gg) of length 4.

An object of class GeomSerialaxes (inherits from GeomPath, Geom, ggproto, gg) of length 3.

---

geom_density_ *More general smoothed density estimates*

Description

Computes and draws kernel density estimate. Compared with geom_density(), it provides more general cases that accepting x and y. See details

Usage

```
geom_density_(
  mapping = NULL,
  data = NULL,
  stat = "density_",
  position = "identity_",
  ...,
  scale.x = NULL,
  scale.y = c("data", "group", "variable"),
  as.mix = FALSE,
  positive = TRUE,
  prop = 0.9,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_density_(
  mapping = NULL,
  data = NULL,
  geom = "density_",
  position = "stack_",
  ...,
  bw = "nrd0",
  adjust = 1,
  kernel = "gaussian",
  n = 512,
  trim = FALSE,
  na.rm = FALSE,
```

```
  orientation = NA,
  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.
...	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.

`scale.x` A sorted length 2 numerical vector representing the range of the whole data will be scaled to. The default value is (0, 1).

`scale.y` one of `data` and `group` to specify.

Type	Description
data (default)	The density estimates are scaled by the whole data set
group	The density estimates are scaled by each group

If the `scale.y` is `data`, it is meaningful to compare the density (shape and area) across all groups; else it is only meaningful to compare the density within each group. See details.

`as.mix` Logical. Within each group, if `TRUE`, the sum of the density estimate area is mixed and scaled to maximum 1. The area of each subgroup (in general, within each group one color represents one subgroup) is proportional to the count; if `FALSE` the area of each subgroup is the same, with maximum 1. See details.

`positive` If `y` is set as the density estimate, where the smoothed curved is faced to, right ('positive') or left ('negative') as vertical layout; up ('positive') or down ('negative') as horizontal layout?

`prop` adjust the proportional maximum height of the estimate (density, histogram, ...).

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

`orientation` The orientation of the layer. The default (`NA`) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be given explicitly by setting `orientation` to either `"x"` or `"y"`. See the *Orientation* section for more detail.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. 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, stat` Use to override the default connection between `geom_density()` and `stat_density()`. For more information about overriding these connections, see how the stat and geom arguments work.

`bw` The smoothing bandwidth to be used. If numeric, the standard deviation of the smoothing kernel. If character, a rule to choose the bandwidth, as listed in `stats::bw.nrd()`. Note that automatic calculation of the bandwidth does not take weights into account.

adjust	A multiplicate bandwidth adjustment. 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.
kernel	Kernel. See list of available kernels in density().
n	number of equally spaced points at which the density is to be estimated, should be a power of two, see density() for details
trim	If FALSE, the default, each density is computed on the full range of the data. If TRUE, each density is computed over the range of that group: this typically means the estimated x values will not line-up, and hence you won't be able to stack density values. This parameter only matters if you are displaying multiple densities in one plot or if you are manually adjusting the scale limits.

Details

The x (or y) is a group variable (categorical) and y (or x) is the target variable (numerical) to be plotted. If only one of x or y is provided, it will treated as a target variable and ggplot2::geom_density will be executed.

There are four combinations of scale.y and as.mix.

scale.y = **"group" and** as.mix = **FALSE** The density estimate area of each subgroup (represented by each color) within the same group is the same.

scale.y = **"group" and** as.mix = **TRUE** The density estimate area of each subgroup (represented by each color) within the same group is proportional to its own counts.

scale.y = **"data" and** as.mix = **FALSE** The sum of density estimate area of all groups is scaled to maximum of 1. and the density area for each group is proportional to the its count. Within each group, the area of each subgroup is the same.

scale.y = **"data" and** as.mix = **TRUE** The sum of density estimate area of all groups is scaled to maximum of 1 and the area of each subgroup (represented by each color) is proportional to its own count.

See vignettes[https://great-northern-diver.github.io/ggmulti/articles/histogram-density-.html] for more intuitive explanation.

Orientation

This geom treats each axis differently and, thus, can thus have two orientations. Often the orientation is easy to deduce from a combination of the given mappings and the types of positional scales in use. Thus, ggplot2 will by default try to guess which orientation the layer should have. Under rare circumstances, the orientation is ambiguous and guessing may fail. In that case the orientation can be specified directly using the orientation parameter, which can be either "x" or "y". The value gives the axis that the geom should run along, "x" being the default orientation you would expect for the geom.

See Also

geom_density, geom_hist_

Examples

```
if(require(dplyr)) {
  mpg %>%
    dplyr::filter(drv != "f") %>%
    ggplot(mapping = aes(x = drv, y = cty, fill = factor(cyl))) +
    geom_density_(alpha = 0.1)

  # only `x` or `y` is provided
  # that would be equivalent to call function `geom_density()`
  diamonds %>%
    dplyr::sample_n(500) %>%
    ggplot(mapping = aes(x = price)) +
    geom_density_()

  # density and boxplot
  # set the density estimate on the left
  mpg %>%
    dplyr::filter(drv != "f") %>%
    ggplot(mapping = aes(x = drv, y = cty,
                         fill = factor(cyl))) +
    geom_density_(alpha = 0.1,
                  scale.y = "group",
                  as.mix = FALSE,
                  positive = FALSE) +
    geom_boxplot()

  # x as density
  set.seed(12345)
  suppressWarnings(
    diamonds %>%
      dplyr::sample_n(500) %>%
      ggplot(mapping = aes(x = price, y = cut, fill = color)) +
      geom_density_(orientation = "x", prop = 0.25,
                    position = "stack_",
                    scale.y = "group")
  )
}
# settings of `scale.y` and `as.mix`

ggplots <- lapply(list(
                      list(scale.y = "data", as.mix = TRUE),
                      list(scale.y = "data", as.mix = FALSE),
                      list(scale.y = "group", as.mix = TRUE),
                      list(scale.y = "group", as.mix = FALSE)
                    ),
                    function(vars) {
                      scale.y <- vars[["scale.y"]]
                      as.mix <- vars[["as.mix"]]
                      ggplot(mpg,
                             mapping = aes(x = drv, y = cty, fill = factor(cyl))) +
                        geom_density_(alpha = 0.1, scale.y = scale.y, as.mix = as.mix) +
                        labs(title = paste("scale.y =", scale.y),
```

```
                            subtitle = paste("as.mix =", as.mix))
                  })
suppressWarnings(
  gridExtra::grid.arrange(grobs = ggplots)
)
```

---

geom_hist_ *More general histogram*

Description

More general histogram (geom_histogram) or bar plot (geom_bar). Both x and y could be accommodated. See details

Usage

```
geom_hist_(
  mapping = NULL,
  data = NULL,
  stat = "hist_",
  position = "stack_",
  ...,
  scale.x = NULL,
  scale.y = c("data", "group", "variable"),
  as.mix = FALSE,
  binwidth = NULL,
  bins = NULL,
  positive = TRUE,
  prop = 0.9,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_histogram_(
  mapping = NULL,
  data = NULL,
  stat = "bin_",
  position = "stack_",
  ...,
  scale.x = NULL,
  scale.y = c("data", "group"),
  as.mix = FALSE,
  positive = TRUE,
  prop = 0.9,
```

```
  binwidth = NULL,
  bins = NULL,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_bar_(
  mapping = NULL,
  data = NULL,
  stat = "count_",
  position = "stack_",
  ...,
  scale.x = NULL,
  scale.y = c("data", "group"),
  positive = TRUE,
  prop = 0.9,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_hist_(
  mapping = NULL,
  data = NULL,
  geom = "bar_",
  position = "stack_",
  ...,
  binwidth = NULL,
  bins = NULL,
  center = NULL,
  boundary = NULL,
  breaks = NULL,
  closed = c("right", "left"),
  pad = FALSE,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_bin_(
  mapping = NULL,
  data = NULL,
  geom = "bar_",
  position = "stack_",
```

```
  ...,
  binwidth = NULL,
  bins = NULL,
  center = NULL,
  boundary = NULL,
  breaks = NULL,
  closed = c("right", "left"),
  pad = FALSE,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_count_(
  mapping = NULL,
  data = NULL,
  geom = "bar_",
  position = "stack_",
  ...,
  na.rm = FALSE,
  orientation = NA,
  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	Position adjustment, either as a string, or the result of a call to a position adjustment function. Function geom_hist_ and geom_histogram_ understand stack_ (stacks bars on top of each other), or dodge_ and dodge2_ (overlapping objects side-to-side) instead of stack, dodge or dodge2
...	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.

`scale.x` A sorted length 2 numerical vector representing the range of the whole data will be scaled to. The default value is (0, 1).

`scale.y` one of `data` and `group` to specify.

Type	Description
data (default)	The density estimates are scaled by the whole data set
group	The density estimates are scaled by each group

If the `scale.y` is `data`, it is meaningful to compare the density (shape and area) across all groups; else it is only meaningful to compare the density within each group. See details.

`as.mix` Logical. Within each group, if `TRUE`, the sum of the density estimate area is mixed and scaled to maximum 1. The area of each subgroup (in general, within each group one color represents one subgroup) is proportional to the count; if `FALSE` the area of each subgroup is the same, with maximum 1. See details.

`binwidth` The width of the bins. Can be specified as a numeric value or as a function that takes x after scale transformation as input and returns a single numeric value. When specifying a function along with a grouping structure, the function will be called once per group. The default is to use the number of bins in `bins`, covering the range of the data. You should always override this value, exploring multiple widths to find the best to illustrate the stories in your data.

The bin width of a date variable is the number of days in each time; the bin width of a time variable is the number of seconds.

`bins` Number of bins. Overridden by `binwidth`. Defaults to 30.

`positive` If `y` is set as the density estimate, where the smoothed curved is faced to, right ('positive') or left ('negative') as vertical layout; up ('positive') or down ('negative') as horizontal layout?

`prop` adjust the proportional maximum height of the estimate (density, histogram, ...).

na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
orientation	The orientation of the layer. The default (NA) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be given explicitly by setting orientation to either "x" or "y". See the *Orientation* section for more detail.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. 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, stat	Use to override the default connection between geom_hist_()/geom_histogram_()/geom_bar_() and stat_hist_()/stat_bin_()/stat_count_().
center, boundary	bin position specifiers. Only one, center or boundary, may be specified for a single plot. center specifies the center of one of the bins. boundary specifies the boundary between two bins. Note that if either is above or below the range of the data, things will be shifted by the appropriate integer multiple of binwidth. For example, to center on integers use binwidth = 1 and center = 0, even if 0 is outside the range of the data. Alternatively, this same alignment can be specified with binwidth = 1 and boundary = 0.5, even if 0.5 is outside the range of the data.
breaks	Alternatively, you can supply a numeric vector giving the bin boundaries. Overrides binwidth, bins, center, and boundary. Can also be a function that takes group-wise values as input and returns bin boundaries.
closed	One of "right" or "left" indicating whether right or left edges of bins are included in the bin.
pad	If TRUE, adds empty bins at either end of x. This ensures frequency polygons touch 0. Defaults to FALSE.

Details

x (or y) is a group variable (categorical) and y (or x) a target variable (numerical) to be plotted. If only one of x or y is provided, it will treated as a target variable and ggplot2::geom_histogram will be executed. Several things should be noticed:

1. If both x and y are given, they can be one discrete one continuous or two discrete. But they cannot be two continuous variables (which one will be considered as a group variable?).

2. geom_hist_ is a wrapper of geom_histogram_ and geom_count_. Suppose the y is our interest (x is the categorical variable), geom_hist_() can accommodate either continuous or discrete y. While, geom_histogram_() only accommodates the continuous y and geom_bar_() only accommodates the discrete y.

3. There are four combinations of scale.y and as.mix.

`scale.y` = **"group" and** `as.mix` = **FALSE** The density estimate area of each subgroup (represented by each color) within the same group is the same.

`scale.y` = **"group" and** `as.mix` = **TRUE** The density estimate area of each subgroup (represented by each color) within the same group is proportional to its own counts.

`scale.y` = **"data" and** `as.mix` = **FALSE** The sum of density estimate area of all groups is scaled to maximum of 1. and the density area for each group is proportional to the its count. Within each group, the area of each subgroup is the same.

`scale.y` = **"data" and** `as.mix` = **TRUE** The sum of density estimate area of all groups is scaled to maximum of 1 and the area of each subgroup (represented by each color) is proportional to its own count.

See vignettes[https://great-northern-diver.github.io/ggmulti/articles/histogram-density-.html] for more intuitive explanation. Note that, if it is a grouped bar chart (both `x` and `y` are categorical), parameter 'as.mix' is meaningless.

Orientation

This geom treats each axis differently and, thus, can thus have two orientations. Often the orientation is easy to deduce from a combination of the given mappings and the types of positional scales in use. Thus, ggplot2 will by default try to guess which orientation the layer should have. Under rare circumstances, the orientation is ambiguous and guessing may fail. In that case the orientation can be specified directly using the `orientation` parameter, which can be either `"x"` or `"y"`. The value gives the axis that the geom should run along, `"x"` being the default orientation you would expect for the geom.

See Also

`geom_histogram`, `geom_density_`

Examples

```
if(require(dplyr) && require(tidyr)) {

  # histogram
  p0 <- mpg %>%
    dplyr::filter(manufacturer %in% c("dodge", "ford", "toyota", "volkswagen")) %>%
    ggplot(mapping = aes(x = manufacturer, y = cty))
  p0 + geom_hist_()

  ## set position
  #### default is "stack_"
  p0 + geom_hist_(mapping = aes(fill = fl))
  #### "dodge_"
  p0 + geom_hist_(position = "dodge_",
                  mapping = aes(fill = fl))
  #### "dodge2_"
  p0 + geom_hist_(position = "dodge2_",
                  mapping = aes(fill = fl))

  # bar chart
```

```
  mpg %>%
    ggplot(mapping = aes(x = drv, y = class)) +
    geom_hist_(orientation = "y")

  # scale.y as "group"
  p <- iris %>%
    tidyr::pivot_longer(cols = -Species,
                        names_to = "Outer sterile whorls",
                        values_to = "x") %>%
    ggplot(mapping = aes(x = `Outer sterile whorls`,
                         y = x, fill = Species)) +
    stat_hist_(scale.y = "group",
               prop = 0.6,
               alpha = 0.5)
  p
  # with density on the left
  p + stat_density_(scale.y = "group",
                    prop = 0.6,
                    alpha = 0.5,
                    positive = FALSE)

  ############ only `x` or `y` is provided ############
  # that would be equivalent to call function
  # `geom_histogram()` or `geom_bar()`
  ### histogram
  diamonds %>%
    dplyr::sample_n(500) %>%
    ggplot(mapping = aes(x = price)) +
    geom_hist_()
  ### bar chart
  diamonds %>%
    dplyr::sample_n(500) %>%
    ggplot(mapping = aes(x = cut)) +
    geom_hist_()
}
```

`geom_image_glyph` *Add image glyphs on scatter plot*

Description

Each point glyph can be an image (png, jpeg, etc) object.

Usage

```
geom_image_glyph(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
```

```
  ...,
  images,
  imagewidth = 1.2,
  imageheight = 0.9,
  interpolate = 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.

images — a list of images (a raster object, bitmap image). If not provided, a point visual (geom_point()) will be displayed.

imagewidth — Numerical; width of image

imageheight — Numerical; height of image

interpolate — A logical value indicating whether to linearly interpolate the image (the alternative is to use nearest-neighbour interpolation, which gives a more blocky result). See rasterGrob.

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

a geom layer

Aesthetics

geom_..._glyph() understands the following aesthetics (required aesthetics are in bold):

- **x**

- **y**
- alpha
- colour
- fill
- group
- size
- linetype
- shape
- stroke

The size unit is `cm`

Note that the shape and stroke do not have real meanings unless the essential argument `images` is missing. If so, a point visual will be displayed with corresponding shape and stroke.

See Also

`geom_serialaxes_glyph`, `geom_polygon_glyph`

Examples

```
# image glyph
if(require("png")) {
img_path <- list.files(file.path(find.package(package = 'ggmulti'),
                                 "images"),
                       full.names = TRUE)
Raptors <- png::readPNG(img_path[2L])
Warriors <- png::readPNG(img_path[3L])

pg <- ggplot(data = data.frame(x = 1:2, y = rep(1, 2)),
       mapping = aes(x = x, y = y)) +
  geom_image_glyph(images = list(Raptors,
                                 Warriors),
                   imagewidth = rep(1.2, 2),
                   imageheight = c(0.9, 1.2)) +
  coord_cartesian(xlim = extendrange(c(1,2)))
pg
# query the images (a numerical array)
build <- ggplot2::ggplot_build(pg)
# `imageRaptors` and `imageWarriors` are three dimensional
# arrays (third dimension specifying the plane)
imageRaptors <- build$data[[1]]$images[[1]]
imageWarriors <- build$data[[1]]$images[[2]]

if(require("grid")) {
grid.newpage()
grid.raster(imageRaptors)
grid.newpage()
grid.raster(imageWarriors)
}
```

```
# THIS IS SLOW
mercLogo <- png::readPNG(img_path[1L])

p <- ggplot(mapping = aes(x = hp, y = mpg)) +
       geom_point(
         data = mtcars[!grepl("Merc", rownames(mtcars)), ],
         color = "skyblue") +
       geom_image_glyph(
         data = mtcars[grepl("Merc", rownames(mtcars)), ],
         images = mercLogo,
         imagewidth = 1.5
       )
p

}
```

`geom_polygon_glyph` *Add polygon glyphs on scatter plot*

Description

Each point glyph can be a polygon object. We provide some common polygon coords in `polygon_glyph`. Also, users can customize their own polygons.

Usage

```
geom_polygon_glyph(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  polygon_x,
  polygon_y,
  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)`).

`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.

`polygon_x` nested list of x-coordinates of polygons, one list element for each scatterplot point. If not provided, a point visual (`geom_point()`) will be displayed.

`polygon_y` nested list of y-coordinates of polygons, one list element for each scatterplot point. If not provided, a point visual (`geom_point()`) will be displayed.

`linewidth` line width of the "glyph" object

`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

a `geom` layer

Aesthetics

geom_..._glyph() understands the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- alpha
- colour
- fill
- group
- size
- linetype
- shape
- stroke

The size unit is `cm`

Note that the shape and stroke do not have real meanings unless the essential argument `polygon_x` or `polygon_y` is missing. If so, a point visual will be displayed with corresponding shape and stroke.

See Also

geom_serialaxes_glyph, geom_image_glyph

Examples

```
# polygon glyph
p <- ggplot(data = data.frame(x = 1:4, y = 1:4),
            mapping = aes(x = x, y = y)) +
  geom_polygon_glyph(polygon_x = list(x_star, x_cross, x_hexagon, x_airplane),
                     polygon_y = list(y_star, y_cross, y_hexagon, y_airplane),
                     colour = 'black', fill = 'red')
p

# the coords of each polygons can be achieved by calling function `ggplot_build`
build <- ggplot2::ggplot_build(p)
polygon_x <- build$data[[1]]$polygon_x
polygon_y <- build$data[[1]]$polygon_y
```

---

geom_quantiles *Add quantile layers on serial axes coordinate*

Description

In ggplot2, geom_quantile() is used to fit a quantile regression to the data and draws the fitted quantiles with lines. However, geom_quantiles() is mainly used to draw quantile lines on serial axes. See examples

Usage

```
geom_quantiles(
  mapping = NULL,
  data = NULL,
  stat = "quantile",
  position = "identity",
  ...,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  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, as a string.
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.

lineend	Line end style (round, butt, square).
linejoin	Line join style (round, mitre, bevel).
linemitre	Line mitre limit (number greater than 1).
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().

See Also

geom_serialaxes_quantile

Examples

```
p <- ggplot(iris,
            mapping = aes(
              Sepal.Length = Sepal.Length,
              Sepal.Width = Sepal.Width,
              Petal.Length = Petal.Length,
              Petal.Width = Petal.Width
            )
  ) +
  geom_path(alpha = 0.2)  +
  coord_serialaxes(scaling = "variable")
p + geom_quantiles(colour = c("red", "green", "blue"),
                   quantiles = c(0.25, 0.5, 0.75),
                   linewidth = 2)
```

---

geom_serialaxes *Serial axes layer*

Description

Draw a serial axes layer, parallel axes under Cartesian system and radial axes under Polar system. It only takes the "widens" data. Each non-aesthetics component defined in the mapping aes() will be treated as an axis.

Usage

```
geom_serialaxes(
  mapping = NULL,
  data = NULL,
  stat = "serialaxes",
  position = "identity",
  ...,
  axes.sequence = character(0L),
  merge = TRUE,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_serialaxes(
  mapping = NULL,
  data = NULL,
  geom = "serialaxes",
  position = "identity",
  ...,
  axes.sequence = character(0L),
  merge = TRUE,
  axes.position = NULL,
  scaling = c("data", "variable", "observation", "none"),
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_dotProduct(
  mapping = NULL,
  data = NULL,
  geom = "path",
  position = "identity",
  ...,
  axes.sequence = character(0L),
  merge = TRUE,
  scaling = c("data", "variable", "observation", "none"),
  transform = andrews,
  na.rm = FALSE,
  orientation = NA,
  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.

`axes.sequence` A vector to define the axes sequence. In serial axes coordinate, the sequence can be either determined in `mapping` (function `aes()`) or by `axes.sequence`. The only difference is that the `mapping` aesthetics will omit the duplicated axes (check examples in `geom_serialaxes`).

`merge` Should `axes.sequence` be merged with mapping aesthetics as a single mapping `uneval` object?

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

`orientation` The orientation of the layer. The default (`NA`) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be given explicitly by setting `orientation` to either `"x"` or `"y"`. See the *Orientation* section for more detail.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. 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.

`axes.position` A numerical vector to determine the axes sequence position; the length should be the same with the length of `axes.sequence` (or mapping `aesthetics`, see examples).

`scaling` one of `data`, `variable`, `observation` or `none` (not suggested the layout is the same with `data`) to specify how the data is scaled.

`transform` A transformation function, can be either `andrews`, `legendre` or some other customized transformation functions.

Details

The difference between the "lengthens" data and "widens" data can be found in Tidy Data. How to transform one to the other is explained in tidyr

See Also

`coord_serialaxes`, `geom_serialaxes_density`, `geom_serialaxes_quantile`, `geom_serialaxes_hist`

Andrews plot `andrews`, Legendre polynomials `legendre`

Examples

```
# parallel coordinate
p <- ggplot(NBAstats2021,
            mapping = aes(FGA = FGA,
                          `3PA` = `3PA`,
                          FTA = FTA,
                          OFGA = OFGA,
                          O3PA = O3PA,
                          OFTA = OFTA,
                          colour = CONF))

# Teams in West are more likely to make 3-point field goals.
# Besides, they have a better performance in restricting opponents
# to make 3-point field goals.
p +
  geom_serialaxes(scaling = "variable",
                  alpha = 0.4,
                  linewidth = 3) +
  scale_x_continuous(breaks = 1:6,
                     labels = c("FGA", "3PA", "FTA",
                                "OFGA", "O3PA", "OFTA")) +
  scale_y_continuous(labels = NULL)

# andrews plot
p + geom_serialaxes(stat = "dotProduct",
                    scaling = "variable",
                    transform = andrews) # default

# Legendre polynomials
p + geom_serialaxes(stat = "dotProduct",
                    scaling = "variable",
                    transform = legendre)


############## Determine axes sequence
# 1. set the duplicated axes by mapping aesthetics
ggplot(iris, mapping = aes(Sepal.Length = Sepal.Length,
                           Sepal.Width = Sepal.Width,
```

```
                          Sepal.Length = Sepal.Length,
                          Sepal.Width = Sepal.Width,
                          colour = Species)) +
  # only two axes, duplicated axes are removed
  geom_serialaxes()

# 2. set the duplicated axes by axes.sequence
ggplot(iris, mapping = aes(colour = Species)) +
  geom_serialaxes(
    axes.sequence = c("Sepal.Length", "Sepal.Width",
                      "Sepal.Length", "Sepal.Width"))
```

`geom_serialaxes_density` *Smoothed density estimates for "widens" data under serial axes coordinate*

Description

Computes and draws kernel density estimates on serial axes coordinate for each non-aesthetics component defined in the mapping `aes()`.

Usage

```
geom_serialaxes_density(
  mapping = NULL,
  data = NULL,
  stat = "serialaxes_density",
  position = "identity_",
  ...,
  axes.sequence = character(0L),
  merge = TRUE,
  scale.y = c("data", "group"),
  as.mix = TRUE,
  positive = TRUE,
  prop = 0.9,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_serialaxes_density(
  mapping = NULL,
  data = NULL,
  geom = "serialaxes_density",
  position = "stack_",
```

```
  ...,
  axes.sequence = character(0L),
  merge = TRUE,
  axes.position = NULL,
  scaling = c("data", "variable", "observation", "none"),
  bw = "nrd0",
  adjust = 1,
  kernel = "gaussian",
  n = 512,
  trim = FALSE,
  na.rm = FALSE,
  orientation = NA,
  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.

`axes.sequence` A vector to define the axes sequence. In serial axes coordinate, the sequence can be either determined in `mapping` (function `aes()`) or by `axes.sequence`. The only difference is that the `mapping` aesthetics will omit the duplicated axes (check examples in `geom_serialaxes`).

`merge` Should `axes.sequence` be merged with mapping aesthetics as a single mapping `uneval` object?

`scale.y` one of `data` and `group` to specify.

Type	Description
data (default)	The density estimates are scaled by the whole data set
group	The density estimates are scaled by each group

If the `scale.y` is `data`, it is meaningful to compare the density (shape and area) across all groups; else it is only meaningful to compare the density within each group. See details.

`as.mix` Logical. Within each group, if `TRUE`, the sum of the density estimate area is mixed and scaled to maximum 1. The area of each subgroup (in general, within each group one color represents one subgroup) is proportional to the count; if `FALSE` the area of each subgroup is the same, with maximum 1. See details.

positive — If y is set as the density estimate, where the smoothed curved is faced to, right ('positive') or left ('negative') as vertical layout; up ('positive') or down ('negative') as horizontal layout?

prop — adjust the proportional maximum height of the estimate (density, histogram, ...).

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

orientation — The orientation of the layer. The default (NA) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be given explicitly by setting orientation to either "x" or "y". See the *Orientation* section for more detail.

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. 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.

axes.position — A numerical vector to determine the axes sequence position; the length should be the same with the length of axes.sequence (or mapping aesthetics, see examples).

scaling — one of data, variable, observation or none (not suggested the layout is the same with data) to specify how the data is scaled.

bw — The smoothing bandwidth to be used. If numeric, the standard deviation of the smoothing kernel. If character, a rule to choose the bandwidth, as listed in stats::bw.nrd(). Note that automatic calculation of the bandwidth does not take weights into account.

adjust — A multiplicate bandwidth adjustment. 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.

kernel — Kernel. See list of available kernels in density().

n — number of equally spaced points at which the density is to be estimated, should be a power of two, see density() for details

trim — If FALSE, the default, each density is computed on the full range of the data. If TRUE, each density is computed over the range of that group: this typically means the estimated x values will not line-up, and hence you won't be able to stack density values. This parameter only matters if you are displaying multiple densities in one plot or if you are manually adjusting the scale limits.

See Also

geom_density_, geom_serialaxes, geom_serialaxes_quantile, geom_serialaxes_hist

Examples

```
p <- ggplot(iris, mapping = aes(Sepal.Length = Sepal.Length,
                                Sepal.Width = Sepal.Width,
                                Petal.Length = Petal.Length,
                                Petal.Width = Petal.Width,
                                colour = Species,
                                fill = Species)) +
       geom_serialaxes(alpha = 0.2) +
       geom_serialaxes_density(alpha = 0.5) +
       scale_x_continuous(breaks = 1:4,
                          labels = colnames(iris)[-5]) +
       scale_y_continuous(labels = NULL) +
       xlab("variable") +
       ylab("") +
       theme(axis.text.x = element_text(angle = 45, vjust = 0.5))
p
```

---

geom_serialaxes_glyph *Add serial axes glyphs on scatter plot*

Description

To visualize high dimensional data on scatterplot. Each point glyph is surrounded by a serial axes (parallel axes or radial axes) object.

Usage

```
geom_serialaxes_glyph(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  serialaxes.data,
  axes.sequence = character(0L),
  scaling = c("data", "variable", "observation", "none"),
  axes.layout = c("parallel", "radial"),
  andrews = FALSE,
```

```
  show.axes = FALSE,
  show.enclosing = FALSE,
  linewidth = 1,
  axescolour = "black",
  bboxcolour = "black",
  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.

serialaxes.data
: a serial axes numerical data set. If not provided, a point visual (geom_point()) will be displayed.

axes.sequence
: A vector to define the axes sequence. In serial axes coordinate, the sequence can be either determined in mapping (function aes()) or by axes.sequence. The only difference is that the mapping aesthetics will omit the duplicated axes (check examples in geom_serialaxes).

scaling
: one of data, variable, observation or none (not suggested the layout is the same with data) to specify how the data is scaled.

axes.layout
: either "radial" or "parallel"

andrews
: Logical; Andrew's plot (a Fourier transformation)

show.axes
: boolean to indicate whether axes should be shown or not

show.enclosing
: boolean to indicate whether enclosing should be shown or not

linewidth
: line width of the "glyph" object

axescolour
: axes color

bboxcolour
: bounding box color

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

a geom layer

Aesthetics

geom_..._glyph() understands the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- alpha
- colour
- fill
- group
- size
- linetype
- shape
- stroke

The size unit is cm

Note that the shape and stroke do not have real meanings unless the essential argument serialaxes.data is missing. If so, a point visual will be displayed with corresponding shape and stroke.

See Also

geom_polygon_glyph, geom_image_glyph

Examples

```
# serial axes glyph
p <- ggplot(data = iris,
            mapping = aes(x = Sepal.Length, y = Sepal.Width, colour = Species)) +
  geom_serialaxes_glyph(serialaxes.data = iris[, -5],
                        axes.layout = "radial")
p
```

---

geom_serialaxes_hist *Histogram for "widens" data under serial axes coordinate*

Description

Computes and draws histogram on serial axes coordinate for each non-aesthetics component defined in the mapping aes().

Usage

```
geom_serialaxes_hist(
  mapping = NULL,
  data = NULL,
  stat = "serialaxes_hist",
  position = "stack_",
  ...,
  axes.sequence = character(0L),
  axes.position = NULL,
  merge = TRUE,
  scale.y = c("data", "group"),
  as.mix = TRUE,
  positive = TRUE,
  prop = 0.9,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_serialaxes_hist(
  mapping = NULL,
  data = NULL,
  geom = "serialaxes_hist",
  position = "stack_",
  ...,
  axes.sequence = character(0L),
  scaling = c("data", "variable", "observation", "none"),
  axes.position = NULL,
  binwidth = NULL,
  bins = NULL,
  center = NULL,
  boundary = NULL,
  breaks = NULL,
  closed = c("right", "left"),
  pad = FALSE,
  na.rm = FALSE,
  orientation = NA,
  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.

`axes.sequence` A vector to define the axes sequence. In serial axes coordinate, the sequence can be either determined in `mapping` (function `aes()`) or by `axes.sequence`. The only difference is that the `mapping` aesthetics will omit the duplicated axes (check examples in `geom_serialaxes`).

`axes.position` A numerical vector to determine the axes sequence position; the length should be the same with the length of `axes.sequence` (or mapping `aesthetics`, see examples).

`merge` Should `axes.sequence` be merged with mapping aesthetics as a single mapping `uneval` object?

`scale.y` one of `data` and `group` to specify.

Type	Description
data (default)	The density estimates are scaled by the whole data set
group	The density estimates are scaled by each group

If the `scale.y` is `data`, it is meaningful to compare the density (shape and area) across all groups; else it is only meaningful to compare the density within each group. See details.

`as.mix` Logical. Within each group, if `TRUE`, the sum of the density estimate area is mixed and scaled to maximum 1. The area of each subgroup (in general, within each group one color represents one subgroup) is proportional to the count; if `FALSE` the area of each subgroup is the same, with maximum 1. See details.

`positive` If `y` is set as the density estimate, where the smoothed curved is faced to, right ('positive') or left ('negative') as vertical layout; up ('positive') or down ('negative') as horizontal layout?

`prop` adjust the proportional maximum height of the estimate (density, histogram, ...).

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

`orientation` The orientation of the layer. The default (`NA`) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be given explicitly by setting `orientation` to either `"x"` or `"y"`. See the *Orientation* section for more detail.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. 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.

`scaling` one of `data`, `variable`, `observation` or `none` (not suggested the layout is the same with `data`) to specify how the data is scaled.

`binwidth` The width of the bins. Can be specified as a numeric value or as a function that takes x after scale transformation as input and returns a single numeric value. When specifying a function along with a grouping structure, the function will be called once per group. The default is to use the number of bins in `bins`, covering the range of the data. You should always override this value, exploring multiple widths to find the best to illustrate the stories in your data.

The bin width of a date variable is the number of days in each time; the bin width of a time variable is the number of seconds.

`bins` Number of bins. Overridden by `binwidth`. Defaults to 30.

`center, boundary`

bin position specifiers. Only one, `center` or `boundary`, may be specified for a single plot. `center` specifies the center of one of the bins. `boundary` specifies the boundary between two bins. Note that if either is above or below the range of the data, things will be shifted by the appropriate integer multiple of `binwidth`. For example, to center on integers use `binwidth = 1` and `center = 0`, even if `0` is outside the range of the data. Alternatively, this same alignment can be specified with `binwidth = 1` and `boundary = 0.5`, even if `0.5` is outside the range of the data.

`breaks` Alternatively, you can supply a numeric vector giving the bin boundaries. Overrides `binwidth`, `bins`, `center`, and `boundary`. Can also be a function that takes group-wise values as input and returns bin boundaries.

`closed` One of `"right"` or `"left"` indicating whether right or left edges of bins are included in the bin.

`pad` If `TRUE`, adds empty bins at either end of x. This ensures frequency polygons touch 0. Defaults to `FALSE`.

See Also

`geom_hist_`, `geom_serialaxes`, `geom_serialaxes_quantile`, `geom_serialaxes_density`

Examples

```
p <- ggplot(NBAstats2021,
            mapping = aes(`FG%` = `FG%`,
```

```
                        `3P%` = `3P%`,
                        `FT%` = `FT%`,
                        `OFG%` = `OFG%`,
                        `O3P%` = `O3P%`,
                        `OFT%` = `OFT%`,
                        colour = Playoff,
                        fill = Playoff)) +
              geom_serialaxes(alpha = 0.2,
                              scaling = "variable") +
              geom_serialaxes_hist(alpha = 0.5,
                                   prop = 0.7,
                                   scaling = "variable") +
              scale_x_continuous(breaks = 1:6,
                                 labels = c("FG", "3P", "FT",
                                            "OFG", "O3P", "OFT")) +
              scale_y_continuous(labels = NULL) +
              xlab("variable") +
              ylab("") +
              theme(axis.text.x = element_text(angle = 45, vjust = 0.5))
p
```

`geom_serialaxes_quantile` *Quantile layer for serial axes coordinate*

Description

Draw a quantile layer for serial axes coordinate. Don't be confused with `geom_quantile()` which is a quantile regression. See examples.

Usage

```
geom_serialaxes_quantile(
  mapping = NULL,
  data = NULL,
  stat = "serialaxes",
  position = "identity",
  ...,
  quantiles = seq(0, 1, 0.25),
  axes.sequence = character(0L),
  merge = TRUE,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_serialaxes_quantile(
```

```
  mapping = NULL,
  data = NULL,
  geom = "serialaxes_quantile",
  position = "identity",
  ...,
  axes.sequence = character(0L),
  merge = TRUE,
  quantiles = seq(0, 1, 0.25),
  scaling = c("data", "variable", "observation", "none"),
  axes.position = NULL,
  na.rm = FALSE,
  orientation = NA,
  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.

`quantiles` numeric vector of probabilities with values in [0,1]. (Values up to 2e-14 outside that range are accepted and moved to the nearby endpoint.)

`axes.sequence` A vector to define the axes sequence. In serial axes coordinate, the sequence can be either determined in `mapping` (function `aes()`) or by `axes.sequence`. The only difference is that the `mapping` aesthetics will omit the duplicated axes (check examples in `geom_serialaxes`).

`merge` Should `axes.sequence` be merged with mapping aesthetics as a single mapping `uneval` object?

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

`orientation` The orientation of the layer. The default (`NA`) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be given explicitly by setting `orientation` to either `"x"` or `"y"`. See the *Orientation* section for more detail.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. 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.
scaling	one of data, variable, observation or none (not suggested the layout is the same with data) to specify how the data is scaled.
axes.position	A numerical vector to determine the axes sequence position; the length should be the same with the length of axes.sequence (or mapping aesthetics, see examples).

See Also

geom_density_, geom_serialaxes, geom_serialaxes_density, geom_serialaxes_hist

Examples

```
# lower quantile, median and upper quantile
p <- ggplot(iris, mapping = aes(Sepal.Length = Sepal.Length,
                                Sepal.Width = Sepal.Width,
                                Petal.Length = Petal.Length,
                                Petal.Width = Petal.Width)) +
       geom_serialaxes(stat = "dotProduct") +
       geom_serialaxes_quantile(stat = "dotProduct",
                                quantiles = c(0.25, 0.5, 0.75),
                                colour = c("red", "blue", "green"))
p
```

---

get_scaledData *scale data*

Description

It is mainly used in serial axes

Usage

```
get_scaledData(
  data,
  sequence = NULL,
  scaling = c("data", "variable", "observation", "none"),
  displayOrder = NULL,
  reserve = FALSE,
  as.data.frame = FALSE
)
```

Arguments

data	A data frame
sequence	vector with variable names that defines the axes sequence. If NULL, it will be set as the column names automatically.
scaling	one of data, variable, observation or none (not suggested the layout is the same with data) to specify how the data is scaled.
displayOrder	the order of the display
reserve	If TRUE, return the variables not shown in sequence as well; else only return the variables defined in sequence.
as.data.frame	Return a matrix or a data.frame

NBAstats2021 *NBA 30 Teams Statistics in 20-21 Regular Season*

Description

A dataset containing the statistics (e.g. Points Per Game, Average Field Goals Made, etc) of 30 NBA Teams in 2020-2021 regular season

Format

A data frame with 30 rows (teams) and 42 variables:

Team Team Names.

CONF Factor; Conference of Teams (West or East).

DIV Factor; Division of Teams.

Playoff Factor; Whether Teams are in (0 or 1) Playoffs.

PTS Points Per Game.

FGM Average Field Goals Made.

FGA Average Field Goals Attempted.

FG% Field Goal Percentage.

3PM Average 3-Point Field Goals Made.

3PA Average 3-Point Field Goals Attempted.
3P% 3-Point Field Goal Percentage.
FTM Average Free Throws Made.
FTA Average Free Throws Attempted.
FT% Free Throw Percentage.
OR Offensive Rebounds Per Game.
DR Defensive Rebounds Per Game.
REB Rebounds Per Game.
AST Assists Per Game.
STL Steals Per Game.
BLK Blocks Per Game.
TO Turnovers Per Game.
PF Fouls Per Game.
OPTS Opponent Points Per Game.
OFGM Opponent Average Field Goals Made.
OFGA Opponent Average Field Goals Attempted.
OFG% Opponent Field Goal Percentage.
O3PM Opponent Average 3-Point Field Goals Made.
O3PA Opponent Average 3-Point Field Goals Attempted.
O3P% Opponent 3-Point Field Goal Percentage.
OFTM Opponent Average Free Throws Made.
OFTA Opponent Average Free Throws Attempted.
OFT% Opponent Free Throw Percentage.
OOR Opponent Offensive Rebounds Per Game.
ODR Opponent Defensive Rebounds Per Game.
OREB Opponent Rebounds Per Game.
OAST Opponent Assists Per Game.
OSTL Opponent Steals Per Game.
OBLK Opponent Blocks Per Game.
OTO Opponent Turnovers Per Game.
OPF Opponent Fouls Per Game.
Win Win Games in Regular Season.
Lose Loss Games in Regular Season.

Author(s)

Zehao Xu

Source

https://www.espn.com/nba/stats/team/_/season/2021

polygon_glyph *Polygon glyph coordinates*

Description

polygon coordinates scaled to (0, 1)

Usage

```
x_star

y_star

x_cross

y_cross

x_hexagon

y_hexagon

x_airplane

y_airplane

x_maple

y_maple
```

Format

An object of class `numeric` of length 10.
An object of class `numeric` of length 10.
An object of class `numeric` of length 12.
An object of class `numeric` of length 12.
An object of class `numeric` of length 6.
An object of class `numeric` of length 6.
An object of class `numeric` of length 32.
An object of class `numeric` of length 32.
An object of class `numeric` of length 26.
An object of class `numeric` of length 26.

See Also

geom_polygon_glyph

Examples

```
if(require("grid")) {
  library(grid)
  grid.newpage()
  grid.polygon(x=(x_star + 1)/2,
               y=(y_star + 1)/2)
  grid.newpage()
  grid.polygon(x=(x_cross + 1)/2,
               y=(y_cross + 1)/2)
  grid.newpage()
  grid.polygon(x=(x_hexagon + 1)/2,
               y=(y_hexagon + 1)/2)
  grid.newpage()
  grid.polygon(x=(x_airplane + 1)/2,
               y=(y_airplane + 1)/2)
  grid.newpage()
  grid.polygon(x=(x_maple + 1)/2,
               y=(y_maple + 1)/2)
}
```

Position-ggproto *Base Position ggproto classes for ggplot2*

Description

All position_ functions (like position_dodge) return a Position object (like PositionDodge). The Position object is responsible for adjusting the position of overlapping geoms. The way that the position_ functions work is slightly different from the geom_ and stat_ functions, because a position_ function actually "instantiates" the Position object by creating a descendant, and returns that. Each of the Position objects is a ggproto object, descended from the top-level Position.

Usage

```
PositionDodge_

PositionDodge2_

PositionIdentity_

PositionStack_

PositionFill_
```

Format

An object of class PositionDodge_ (inherits from PositionDodge, Position, ggproto, gg) of length 2.

An object of class PositionDodge2_ (inherits from PositionDodge2, PositionDodge, Position, ggproto, gg) of length 2.

An object of class PositionIdentity_ (inherits from PositionIdentity, Position, ggproto, gg) of length 3.

An object of class PositionStack_ (inherits from PositionStack, Position, ggproto, gg) of length 3.

An object of class PositionFill_ (inherits from PositionStack_, PositionStack, Position, ggproto, gg) of length 2.

---

position_dodge_ *Dodge overlapping objects side-to-side*

Description

Dodging preserves the vertical position of an geom while adjusting the horizontal position. position_dodge_() dodges bars side by side but conditional on locations.

Usage

```
position_dodge_(width = NULL, preserve = c("total", "single"))

position_dodge2_(
  width = NULL,
  preserve = c("total", "single"),
  padding = 0.1,
  reverse = FALSE
)
```

Arguments

width	Dodging width, when different to the width of the individual elements. This is useful when you want to align narrow geoms with wider geoms. See the examples.
preserve	Should dodging preserve the "total" width of all elements at a position, or the width of a "single" element?
padding	Padding between elements at the same position. Elements are shrunk by this proportion to allow space between them. Defaults to 0.1.
reverse	If TRUE, will reverse the default stacking order. This is useful if you're rotating both the plot and legend.

Details

It is built based on position_dodge, but used for multiple locations, such as geom_hist_() or geom_density_(). Check examples to see the difference.

Aesthetics

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

- order → NULL

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

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

- order → NULL

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

See Also

See geom_hist_ and geom_serialaxes_hist for more examples.

Other position adjustments for multiple locations: position_identity_, position_stack_, position_fill_

Parent: position_dodge

Examples

```
if(require(dplyr)) {
p <- iris %>%
  tidyr::pivot_longer(cols = -Species,
                      names_to = "Outer sterile whorls",
                      values_to = "values") %>%
  ggplot(data,
         mapping = aes(x = `Outer sterile whorls`,
                       y = values,
                       fill = Species))

p + geom_hist_(position = position_dodge_())
}


# all bins are shifted on the left
p +
  geom_hist_(position = position_dodge())
```

position_identity_ *Don't adjust position*

Description

Don't adjust position

Usage

```
position_identity_()
```

See Also

Other position adjustments for multiple locations: position_stack_, position_fill_, position_dodge_, position_dodge2_

position_stack_ *Stack overlapping objects on top of each another*

Description

position_stack_ stacks bars on top of each other, conditional on locations.

Usage

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

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

Arguments

vjust	Vertical adjustment for geoms that have a position (like points or lines), not a dimension (like bars or areas). Set to 0 to align with the bottom, 0.5 for the middle, and 1 (the default) for the top.
reverse	If TRUE, will reverse the default stacking order. This is useful if you're rotating both the plot and legend.

Details

It is built based on position_stack, but used for multiple locations, such as geom_hist_ or geom_density_. Rather than stack everything on top of each other, position_stack_ stacks bars based on locations. Check examples to see the difference.

See Also

See geom_hist_, geom_density_, geom_serialaxes_density and geom_serialaxes_hist for more examples.

Other position adjustments for multiple locations: position_identity_, position_dodge_, position_dodge2_

Parent: position_stack

Examples

```
p <- ggplot(iris,
       mapping = aes(Sepal.Length = Sepal.Length,
                     Sepal.Width = Sepal.Width,
                     Petal.Length = Petal.Length,
                     Petal.Width = Petal.Width,
                     colour = Species))
p +
 geom_serialaxes_density(position = position_stack_())


p +
  geom_serialaxes_density(position = position_stack())
```

Stat-ggproto *Base Stat ggproto classes for ggplot2*

Description

All stat_ functions (like stat_bin()) return a layer that contains a Stat object (like StatBin). The Stat object is responsible for rendering the data in the plot. Each of the Stat objects is a ggproto object, descended from the top-level Stat, and each implements various methods and fields.

Usage

```
StatDensity_

StatHist_

StatBin_

StatCount_

StatSerialaxesDensity

StatSerialaxesHist
```

`StatSerialaxes`

`StatDotProduct`

Format

An object of class `StatDensity_` (inherits from `StatDensity`, `Stat`, `ggproto`, `gg`) of length 4.

An object of class `StatHist_` (inherits from `StatBin`, `Stat`, `ggproto`, `gg`) of length 4.

An object of class `StatBin_` (inherits from `StatHist_`, `StatBin`, `Stat`, `ggproto`, `gg`) of length 2.

An object of class `StatCount_` (inherits from `StatHist_`, `StatBin`, `Stat`, `ggproto`, `gg`) of length 2.

An object of class `StatSerialaxesDensity` (inherits from `StatDensity`, `Stat`, `ggproto`, `gg`) of length 4.

An object of class `StatSerialaxesHist` (inherits from `StatBin`, `Stat`, `ggproto`, `gg`) of length 4.

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

An object of class `StatDotProduct` (inherits from `StatSerialaxes`, `Stat`, `ggproto`, `gg`) of length 4.

Index