Package ‘ggdist’

October 13, 2022

Title Visualizations of Distributions and Uncertainty
Version 3.2.0
Maintainer Matthew Kay <mjskay@northwestern.edu>

Description
Provides primitives for visualizing distributions using ‘ggplot2’ that are particularly tuned for visualizing uncertainty in either a frequentist or Bayesian mode. Both analytical distributions (such as frequentist confidence distributions or Bayesian priors) and distributions represented as samples (such as bootstrap distributions or Bayesian posterior samples) are easily visualized. Visualization primitives include but are not limited to: points with multiple uncertainty intervals, eye plots (Spiegelhalter D., 1999) <https://ideas.repec.org/a/bla/jorssa/v162y1999i1p45-58.html>, density plots, gradient plots, dot plots (Wilkinson L., 1999) <doi:10.1080/0033358.1999.10474474>, quantile dot plots (Kay M., Kola T., Hullman J., Munson S., 2016) <doi:10.1145/2858036.2858558>, complementary cumulative distribution function barplots (Fernandes M., Walls L., Munson S., Hullman J., Kay M., 2018) <doi:10.1145/3173574.3173718>, and fit curves with multiple uncertainty ribbons.

Depends R (>= 3.5.0)
Imports tidyselect, dplyr (>= 1.0.0), ggplot2 (>= 3.3.5), rlang (>= 0.3.0), scales, grid, HDInterval, tibble, vctrs, withr, distributional (>= 0.3.0), numDeriv, glue
Suggests knitr, testthat, vdiffr (>= 1.0.0), svglite (>= 2.1.0), broom (>= 0.5.6), modelr, cowplot, patchwork, covr, gdtools, rmarkdown, png, fda, forcats, purrr (>= 0.2.3), tidyr (>= 1.0.0), beeswarm (>= 0.4.0), posterior, pkgdown, palmerpenguins

License GPL (>= 3)
Language en-US

BugReports https://github.com/mjskay/ggdist/issues/new
URL. https://mjskay.github.io/ggdist/,
       https://github.com/mjskay/ggdist/

VignetteBuilder knitr
RoxygenNote 7.2.1
LazyData true
Encoding UTF-8

Collate`"ggdist-package.R""util.R""abstract_geom.R"
   "abstract_stat.R""abstract_stat_slabinterval.R"
   "distributions.R""draw_key_slabinterval.R""geom.R"
   "geom_slabinterval.R""geom_dotsinterval.R""geom_interval.R"
   "geom_lineribbon.R""geom_pointinterval.R""geom_slab.R"
   "guide_rampbar.R""lkjcorr_marginal.R""parse_dist.R"
   "point_interval.R""position_dodgejust.R""rd.R"
   "rd_slabinterval.R""rd_lineribbon.R""scale_colour_ramp.R"
   "scale_thickness.R""scale_.R""stat.R""stat_slabinterval.R"
   "stat_dotsinterval.R""stat_pointinterval.R""stat_interval.R"
   "stat_lineribbon.R""student_t.R""testthat.R""theme_ggdist.R"
   "tidy_format_translators.R""deprecated.R"

NeedsCompilation no
Author  Matthew Kay [aut, cre],
       Brenton M. Wiernik [ctb]
Repository CRAN
Date/Publication 2022-07-19 16:40:02 UTC

R topics documented:

ggdist-package .................................................. 3
bin_dots .......................................................... 4
curve_interval .................................................... 6
cut_cdf_qi ........................................................ 9
find_dotplot_binwidth .......................................... 11
geom_dotsinterval ............................................. 12
geom_interval ................................................... 21
geom_lineribbon ............................................... 26
geom_pointinterval ........................................... 29
geom_slab ......................................................... 34
geom_slabinterval .............................................. 38
ggdist-deprecated ............................................... 45
guide_rampbar .................................................... 46
lkjcorr_marginal ................................................. 49
marginalize_lkjcorr ............................................ 51
parse_dist ........................................................ 52
point_interval ................................................... 54
position_dodgejust ............................................. 59
ggdist-package

Visualizations of Distributions and Uncertainty

Description

ggdist is an R package that aims to make it easy to integrate popular Bayesian modeling methods into a tidy data + ggplot workflow.

Details

ggdist is an R package that provides a flexible set of ggplot2 geoms and stats designed especially for visualizing distributions and uncertainty. It is designed for both frequentist and Bayesian uncertainty visualization, taking the view that uncertainty visualization can be unified through the perspective of distribution visualization: for frequentist models, one visualizes confidence distributions or bootstrap distributions (see vignette("freq-uncertainty-vis")); for Bayesian models, one visualizes probability distributions (see vignette("tidybayes", package = "tidybayes")).

The `geom_slabinterval()` / `stat_slabinterval()` family (see vignette("slabinterval")) makes it easy to visualize point summaries and intervals, eye plots, half-eye plots, ridge plots, CCDF bar plots, gradient plots, histograms, and more.

The `geom_dotsinterval()` / `stat_dotsinterval()` family (see vignette("dotsinterval")) makes it easy to visualize dot+interval plots, Wilkinson dotplots, beeswarm plots, and quantile dotplots.

The `geom_lineribbon()` / `stat_lineribbon()` family (see vignette("lineribbon")) makes it easy to visualize fit lines with an arbitrary number of uncertainty bands.
bin_dots  

**Bin data values using a dotplot algorithm**

**Description**

Bins the provided data values using one of several dotplot algorithms.

**Usage**

```
bin_dots(
  x,  
  y,  
  binwidth,  
  heightratio = 1,  
  stackratio = 1,  
  layout = c("bin", "weave", "swarm"),  
  side = c("topright", "top", "right", "bottomleft", "bottom", "left", "topleft",  
    "bottomright", "both"),  
  orientation = c("horizontal", "vertical", "y", "x")
)
```

**Arguments**

- **x**: numeric vector of x values
- **y**: numeric vector of y values
- **binwidth**: bin width
- **heightratio**: ratio of bin width to dot height
- **stackratio**: ratio of dot height to vertical distance between dot centers
- **layout**: The layout method used for the dots:
  - "bin" (default): places dots on the off-axis at the midpoint of their bins as in the classic Wilkinson dotplot. This maintains the alignment of rows and columns in the dotplot. This layout is slightly different from the classic Wilkinson algorithm in that: (1) it nudges bins slightly to avoid overlapping bins and (2) if the input data are symmetrical it will return a symmetrical layout.
  - "weave": uses the same basic binning approach of "bin", but places dots in the off-axis at their actual positions (modulo overlaps, which are nudged out of the way). This maintains the alignment of rows but does not align dots within columns. Does not work well when side = "both".
  - "swarm": uses the "compactswarm" layout from beeswarm::beeswarm(). Does not maintain alignment of rows or columns, but can be more compact and neat looking, especially for sample data (as opposed to quantile dotplots of theoretical distributions, which may look better with "bin" or "weave").

**Description**

Bins the provided data values using one of several dotplot algorithms.
side  Which side to place the slab on. "topright", "top", and "right" are synonyms which cause the slab to be drawn on the top or the right depending on if orientation is "horizontal" or "vertical". "bottomleft", "bottom", and "left" are synonyms which cause the slab to be drawn on the bottom or the left depending on if orientation is "horizontal" or "vertical". "topleft" causes the slab to be drawn on the top or the left, and "bottomright" causes the slab to be drawn on the bottom or the right. "both" draws the slab mirrored on both sides (as in a violin plot).

orientation  Whether the dots are laid out horizontally or vertically. Follows the naming scheme of `geom_slabinterval()`:

- "horizontal" assumes the data values for the dotplot are in the x variable and that dots will be stacked up in the y direction.
- "vertical" assumes the data values for the dotplot are in the y variable and that dots will be stacked up in the x direction.

For compatibility with the base ggplot naming scheme for orientation, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal".

Value  A data.frame with three columns:

- x: the x position of each dot
- y: the y position of each dot
- bin: a unique number associated with each bin (supplied but not used when layout = "swarm")

See Also  

- `find_dotplot_binwidth()` for an algorithm that finds good bin widths to use with this function; `geom_dotsinterval()` for geometries that use these algorithms to create dotplots.

Examples

```
library(dplyr)
library(ggplot2)

x = qnorm(ppoints(20))
bin_df = bin_dots(x = x, y = 0, binwidth = 0.5, heightratio = 1)
bin_df

# we can manually plot the binning above, though this is only recommended
# if you are using find_dotplot_binwidth() and bin_dots() to build your own
grobs. For practical use it is much easier to use geom_dots(), which will
# automatically select good bin widths for you and which uses
# find_dotplot_binwidth() and bin_dots() internally)
bin_df %>%
  ggplot(aes(x = x, y = y)) +
  geom_point(size = 4) +
  coord_fixed()
```
**curve_interval**

*Curvewise point and interval summaries for tidy data frames of draws from distributions*

**Description**

Translates draws from distributions in a grouped data frame into a set of point and interval summaries using a curve boxplot-inspired approach.

**Usage**

```r
curve_interval(
  .data,
  ..., 
  .along = NULL,
  .width = 0.5,
  .interval = c("mhd", "mbd", "bd", "bd-mbd"),
  .simple_names = TRUE,
  na.rm = FALSE,
  .exclude = c(".chain", ".iteration", ".draw", ".row")
)
```

**Arguments**

- **.data**  
  Data frame (or grouped data frame as returned by `group_by()`) that contains draws to summarize.

- **...**  
  Bare column names or expressions that, when evaluated in the context of `.data`, represent draws to summarize. If this is empty, then by default all columns that are not group columns and which are not in `.exclude` (by default ".chain", ".iteration", ".draw", and ".row") will be summarized. This can be list columns.

- **.along**  
  Which columns are the input values to the function describing the curve (e.g., the "x" values). Supports tidyselect syntax, as in `dplyr::select()`. Intervals are calculated jointly with respect to these variables, conditional on all other grouping variables in the data frame. The default (NULL) causes `curve_interval()` to use all grouping variables in the input data frame as the value for `.along`, which will generate the most conservative intervals. However, if you want to calculate intervals for some function `y = f(x)` conditional on some other variable(s) (say, conditional on a factor `g`), you would group by `g`, then use `.along = x` to calculate intervals jointly over `x` conditional on `g`.

- **.width**  
  Vector of probabilities to use that determine the widths of the resulting intervals. If multiple probabilities are provided, multiple rows per group are generated, each with a different probability interval (and value of the corresponding `.width` column).
The method used to calculate the intervals. Currently, all methods rank the curves using some measure of data depth, then create envelopes containing the \(.width\%\) "deepest" curves. Available methods are:

- "mhd": mean halfspace depth (Fraiman and Muniz 2001).
- "mbd": modified band depth (Sun and Genton 2011): calls `fda::fbplot()` with method = "MBD".
- "bd": band depth (Sun and Genton 2011): calls `fda::fbplot()` with method = "BD2".
- "bd-mbd": band depth, breaking ties with modified band depth (Sun and Genton 2011): calls `fda::fbplot()` with method = "Both".

When `TRUE` and only a single column / vector is to be summarized, use the name `.lower` for the lower end of the interval and `.upper` for the upper end. If `.data` is a vector and this is `TRUE`, this will also set the column name of the point summary to `.value`. When `FALSE` and `.data` is a data frame, names the lower and upper intervals for each column `x.lower` and `x.upper`. When `FALSE` and `.data` is a vector, uses the naming scheme `y`, `ymin` and `ymax` (for use with ggplot).

Logical value indicating whether NA values should be stripped before the computation proceeds. If `FALSE` (the default), the presence of NA values in the columns to be summarized will generally result in an error. If `TRUE`, NA values will be removed in the calculation of intervals so long as `.interval` is "mhd"; other methods do not currently support `na.rm`. Be cautious in applying this parameter: in general, it is unclear what a joint interval should be when any of the values are missing!

A character vector of names of columns to be excluded from summarization if no column names are specified to be summarized. Default ignores several meta-data column names used in tidybayes.

Intervals are calculated by ranking the curves using some measure of data depth, then using binary search to find a cutoff \(k\) such that an envelope containing the \(k\%\) "deepest" curves also contains \(.width\%\) of the curves, for each value of \(.width\) (note that \(k\) and \(.width\) are not necessarily the same). This is in contrast to most functional boxplot or curve boxplot approaches, which tend to simply take the \(.width\%\) deepest curves, and are generally quite conservative (i.e. they may contain more than \(.width\%\) of the curves).

See Mirzargar et al. (2014) or Juul et al. (2020) for an accessible introduction to data depth and curve boxplots / functional boxplots.

A data frame containing point summaries and intervals, with at least one column corresponding to the point summary, one to the lower end of the interval, one to the upper end of the interval, the width of the interval (`.width`), the type of point summary (`.point`), and the type of interval (`.interval`).
Author(s)

Matthew Kay

References


See Also

point_interval() for pointwise intervals. See vignette("lineribbon") for more examples and discussion of the differences between pointwise and curvewise intervals.

Examples

```r
library(dplyr)
library(ggplot2)

# generate a set of curves
k = 11 # number of curves
n = 201
df = tibble(.draw = rep(1:k, n),
     .draw = rep(1:k, n),
     mean = rep(seq(-5,5, length.out = k), n),
     x = rep(seq(-15,15,length.out = n), each = k),
     y = dnorm(x, mean, 3)
)

# see pointwise intervals...
df %>%
     group_by(x) %>%
     median_qi(y, .width = c(.5)) %>%
     ggplot(aes(x = x, y = y)) +
     geom_lineribbon(aes(ymin = .lower, ymax = .upper)) +
     geom_line(aes(group = .draw), alpha=0.15, data = df) +
     scale_fill_brewer() +
     ggtitle("50% pointwise intervals with point_interval()") +
     theme_ggdist()

# ... compare them to curvewise intervals
```
**cut_cdf_qi**

```r
df %>%
group_by(x) %>%
curve_interval(y, .width = c(.5)) %>%
ggplot(aes(x = x, y = y)) +
gem_lineribbon(aes(ymin = .lower, ymax = .upper)) +
gem_line(aes(group = .draw), alpha=0.15, data = df) +
scale_fill_brewer() +
ggtitle("50% curvewise intervals with curve_interval()") +
theme_ggdist()
```

---

**cut_cdf_qi**  
*Category values from a CDF into quantile intervals*

**Description**

Given a vector of probabilities from a cumulative distribution function (CDF) and a list of desired quantile intervals, return a vector categorizing each element of the input vector according to which quantile interval it falls into. **NOTE:** While this function can be used for (and was originally designed for) drawing slabs with intervals overlaid on the density, this is can now be done more easily by mapping the `.width` or `level` computed variable to slab fill or color. See **Examples**.

**Usage**

```r
cut_cdf_qi(p, .width = c(0.66, 0.95, 1), labels = NULL)
```

**Arguments**

- `p`  
  A numeric vector of values from a cumulative distribution function, such as values returned by `p`-prefixed distribution functions in base R (e.g. `pnorm()`), the `cdf()` function, or values of the `cdf` computed aesthetic from the `stat_slabinterval()` family of stats.

- `.width`  
  vector of probabilities to use that determine the widths of the resulting intervals.

- `labels`  
  One of:
  - NULL to use the default labels (`.width` converted to a character vector).
  - A character vector giving labels (must be same length as `.width`)
  - A function that takes numeric probabilities as input and returns labels as output (a good candidate might be `scales::percent_format()`).

**Value**

An `ordered` factor of the same length as `p` giving the quantile interval to which each value of `p` belongs.

**See Also**

See `stat_slabinterval()` and its shortcut `stats`, which generate `cdf` aesthetics that can be used with `cut_cdf_qi()` to draw slabs colored by their intervals.
Examples

```r
library(ggplot2)
library(dplyr)
library(scales)
library(distributional)

theme_set(theme_ggdist())

# NOTE: cut_cdf_qi() used to be the recommended way to do intervals overlaid
# on densities, like this...
tibble(x = dist_normal(0, 1)) %>%
  ggplot(aes(xdist = x)) +
  stat_slab(
    aes(fill = stat(cut_cdf_qi(cdf)))
  ) +
  scale_fill_brewer(direction = -1)

# ... however this is now more easily and flexibly accomplished by directly
# mapping .width or level onto fill:
tibble(x = dist_normal(0, 1)) %>%
  ggplot(aes(xdist = x)) +
  stat_slab(
    aes(fill = stat(level)),
    .width = c(.66, .95, 1)
  ) +
  scale_fill_brewer()

# See vignette("slabinterval") for more examples. The remaining examples
# below using cut_cdf_qi() are kept for posterity.

# With a halfeye (or other geom with slab and interval), NA values will
# show up in the fill scale from the CDF function applied to the internal
# interval geometry data and can be ignored, hence na.translate = FALSE
tibble(x = dist_normal(0, 1)) %>%
  ggplot(aes(xdist = x)) +
  stat_halfeye(aes(
    fill = stat(cut_cdf_qi(cdf, .width = c(.5, .8, .95, 1)))
  )) +
  scale_fill_brewer(direction = -1, na.translate = FALSE)

# we could also use the labels parameter to apply nicer formatting
# and provide a better name for the legend, and omit the 100% interval
# if desired
tibble(x = dist_normal(0, 1)) %>%
  ggplot(aes(xdist = x)) +
  stat_halfeye(aes(
    fill = stat(cut_cdf_qi(cdf, .width = c(.5, .8, .95), labels = percent_format(accuracy = 1)))
  )) +
  labs(fill = "Interval") +
  scale_fill_brewer(direction = -1, na.translate = FALSE)
```
find_dotplot_binwidth  

**Find a good bin width for a dotplot**

**Description**

Searches for a nice-looking bin width to use to draw a dotplot such that the height of the dotplot fits within a given space (maxheight).

**Usage**

\[
\text{find_dotplot_binwidth}(x, \text{maxheight}, \text{heightratio} = 1, \text{stackratio} = 1)
\]

**Arguments**

- **x**: numeric vector of values
- **maxheight**: maximum height of the dotplot
- **heightratio**: ratio of bin width to dot height
- **stackratio**: ratio of dot height to vertical distance between dot centers

**Details**

This dynamic bin selection algorithm uses a binary search over the number of bins to find a bin width such that if the input data (x) is binned using a Wilkinson-style dotplot algorithm the height of the tallest bin will be less than maxheight.

This algorithm is used by `geom_dotsinterval()` (and its variants) to automatically select bin widths. Unless you are manually implementing your own dotplot `grob` or `geom`, you probably do not need to use this function directly.

**Value**

A suitable bin width such that a dotplot created with this bin width and heightratio should have its tallest bin be less than or equal to maxheight.

**See Also**

- `bin_dots()` for an algorithm can bin dots using bin widths selected by this function; `geom_dotsinterval()` for geometries that use these algorithms to create dotplots.

**Examples**

```r
library(dplyr)
library(ggplot2)

x = qnorm(ppoints(20))
binwidth = find_dotplot_binwidth(x, maxheight = 4, heightratio = 1)
binwidth
```
bin_df = bin_dots(x = x, y = 0, binwidth = binwidth, heightratio = 1)
bin_df

# we can manually plot the binning above, though this is only recommended
# if you are using find_dotplot_binwidth() and bin_dots() to build your own
# grob. For practical use it is much easier to use geom_dots(), which will
# automatically select good bin widths for you (and which uses
# find_dotplot_binwidth() and bin_dots() internally)
bin_df %>%
  ggplot(aes(x = x, y = y)) +
  geom_point(size = 4) +
  coord_fixed()

---

**geom_dotsinterval**  
**Automatic dotplots, dots + intervals, and quantile dotplots (ggplot geom)**

### Description

Geoms and stats for creating dotplots that automatically determines a bin width that ensures the plot fits within the available space. Also ensures dots do not overlap, and allows generation of quantile dotplots using the quantiles argument to `stat_dotsinterval()`/`stat_dots()`. Generally follows the naming scheme and arguments of the `geom_slabinterval()` and `stat_slabinterval()` family of geoms and stats.

### Usage

```r
geom_dotsinterval(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  binwidth = NA,
  dotsize = 1.07,
  stackratio = 1,
  layout = "bin",
  verbose = FALSE,
  orientation = NA,
  interval_size_domain = c(1, 6),
  interval_size_range = c(0.6, 1.4),
  fatten_point = 1.8,
  show_slab = TRUE,
  show_point = TRUE,
  show_interval = TRUE,
  na.rm = FALSE,
)```
show.legend = NA,
inherit.aes = TRUE
)

dotsinterval()

geom_dots(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ..., 
  binwidth = NA,
  dotsize = 1.07,
  stackratio = 1,
  layout = "bin",
  verbose = FALSE,
  orientation = NA,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_dotsinterval(
  mapping = NULL,
  data = NULL,
  geom = "dotsinterval",
  position = "identity",
  ..., 
  quantiles = NA,
  point_interval = "median_qi",
  .width = c(0.66, 0.95),
  orientation = NA,
  na.rm = FALSE,
  show.legend = c(size = FALSE),
  inherit.aes = TRUE
)

stat_dots(
  mapping = NULL,
  data = NULL,
  geom = "dots",
  position = "identity",
  ..., 
  quantiles = NA,
  orientation = NA,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
**Arguments**

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

**data**
The data to be displayed in this layer. There are three options:
- If NULL, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
- A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
- A function will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x, 10)`).

**stat**
The statistical transformation to use on the data for this layer, as a string.

**position**
Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" (position_dodge()) or "dodgejust" (position_dodgejust()) can be useful if you have overlapping geometries.

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

**binwidth**
The bin width to use for laying out the dots. One of:

- **NA** (the default): Dynamically select the bin width based on the size of the plot when drawn. This will pick a binwidth such that the tallest stack of dots is at most scale in height (ideally exactly scale in height, though this is not guaranteed).
- A length-1 (scalar) numeric or unit object giving the exact bin width.
- A length-2 (vector) numeric or unit object giving the minimum and maximum desired bin width. The bin width will be dynamically selected within these bounds.

If the value is numeric, it is assumed to be in units of data. The bin width (or its bounds) can also be specified using `unit()`, which may be useful if it is desired that the dots be a certain point size or a certain percentage of the width/height of the viewport. For example, `unit(0.1, "npc")` would make dots that are exactly 10% of the viewport size along whichever dimension the dotplot is drawn; `unit(c(0, 0.1), "npc")` would make dots that are at most 10% of the viewport size (while still ensuring the tallest stack is less than or equal to scale).

**dotsize**
The width of the dots relative to the binwidth. The default, 1.07, makes dots be just a bit wider than the bin width, which is a manually-tuned parameter that tends to work well with the default circular shape, preventing gaps between bins from appearing to be too large visually (as might arise from dots being precisely the binwidth). If it is desired to have dots be precisely the binwidth, set dotsize = 1.

**stackratio**
The distance between the center of the dots in the same stack relative to the dot height. The default, 1, makes dots in the same stack just touch each other.
The layout method used for the dots:

- "bin" (default): places dots on the off-axis at the midpoint of their bins as in the classic Wilkinson dotplot. This maintains the alignment of rows and columns in the dotplot. This layout is slightly different from the classic Wilkinson algorithm in that: (1) it nudges bins slightly to avoid overlapping bins and (2) if the input data are symmetrical it will return a symmetrical layout.

- "weave": uses the same basic binning approach of "bin", but places dots in the off-axis at their actual positions (modulo overlaps, which are nudged out of the way). This maintains the alignment of rows but does not align dots within columns. Does not work well when side = "both".

- "swarm": uses the "compactswarm" layout from beeswarm::beeswarm(). Does not maintain alignment of rows or columns, but can be more compact and neat looking, especially for sample data (as opposed to quantile dotplots of theoretical distributions, which may look better with "bin" or "weave").

If TRUE, print out the bin width of the dotplot. Can be useful if you want to start from an automatically-selected bin width and then adjust it manually. Bin width is printed both as data units and as normalized parent coordinates or "npc"s (see unit()). Note that if you just want to scale the selected bin width to fit within a desired area, it is probably easier to use scale than to copy and scale binwidth manually, and if you just want to provide constraints on the bin width, you can pass a length-2 vector to binwidth.

Whether this geom is drawn horizontally or vertically. One of:

- NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.

- "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify different groups. For each group, uses the x, xmin, xmax, and thickness aesthetics to draw points, intervals, and slabs.

- "vertical" (or "x"): draw vertically, using the x aesthetic to identify different groups. For each group, uses the y, ymin, ymax, and thickness aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for orientation, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal" (tidybayes had an orientation parameter before base ggplot did, hence the discrepancy).

A length-2 numeric vector giving the minimum and maximum of the values of the size aesthetic that will be translated into actual sizes for intervals drawn according to interval_size_range (see the documentation for that argument.)

A length-2 numeric vector. This geom scales the raw size aesthetic values when drawing interval and point sizes, as they tend to be too thick when using the default settings of scale_size_continuous(), which give sizes with a range of c(1, 6). The interval_size_domain value indicates the input domain of raw size values (typically this should be equal to the value of the range argument of the scale_size_continuous() function), and interval_size_range
indicates the desired output range of the size values (the min and max of the actual sizes used to draw intervals). Most of the time it is not recommended to change the value of this argument, as it may result in strange scaling of legends; this argument is a holdover from earlier versions that did not have size aesthetics targeting the point and interval separately. If you want to adjust the size of the interval or points separately, you can instead use the interval_size or point_size aesthetics; see scales.

**fatten_point**
A multiplicative factor used to adjust the size of the point relative to the size of the thickest interval line. If you wish to specify point sizes directly, you can also use the point_size aesthetic and scale_point_size_continuous() or scale_point_size_discrete(); sizes specified with that aesthetic will not be adjusted using fatten_point.

**show_slab**
Should the slab portion of the geom be drawn?

**show_point**
Should the point portion of the geom be drawn?

**show_interval**
Should the interval portion of the geom be drawn?

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

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

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

**geom**
Use to override the default connection between stat_slabinterval() and geom_slabinterval()

**quantiles**
Setting this to a value other than NA will produce a quantile dotplot: that is, a dotplot of quantiles from the sample or distribution (for analytical distributions, the default of NA is taken to mean 100 quantiles). The value of quantiles determines the number of quantiles to plot. See Kay et al. (2016) and Fernandes et al. (2018) for more information on quantile dotplots.

**point_interval**
A function from the point_interval() family (e.g., median_qi, mean_qi, mode_hdi, etc), or a string giving the name of a function from that family (e.g., "median_qi", "mean_qi", "mode_hdi", etc; if a string, the caller’s environment is searched for the function, followed by the ggdist environment). This function determines the point summary (typically mean, median, or mode) and interval type (quantile interval, qi; highest-density interval, hdi; or highest-density continuous interval, hdi). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of orientation. See the point_interval() family of functions for more information.

**.width**
The .width argument passed to point_interval: a vector of probabilities to use that determine the widths of the resulting intervals. If multiple probabilities are provided, multiple intervals per group are generated, each with a different probability interval (and value of the corresponding .width and level generated variables).
Details

The dots geoms are similar to `geom_dotplot()` but with a number of differences:

- Dots geoms act like slabs in `geom_slabinterval()` and can be given x positions (or y positions when in a horizontal orientation).
- Given the available space to lay out dots, the dots geoms will automatically determine how many bins to use to fit the available space.
- Dots geoms use a dynamic layout algorithm that lays out dots from the center out if the input data are symmetrical, guaranteeing that symmetrical data results in a symmetrical plot. The layout algorithm also prevents dots from overlapping each other.
- The shape of the dots in a in these geoms can be changed using the `slab_shape` aesthetic (when using the `dotsinterval` family) or the `shape` or `slab_shape` aesthetic (when using the `dots` family).

`stat_dots()` and `stat_dotsinterval()`, when used with the quantiles argument, are particularly useful for constructing quantile dotplots, which can be an effective way to communicate uncertainty using a frequency framing that may be easier for laypeople to understand (Kay et al. 2016, Fernandes et al. 2018).

To visualize sample data, such as a data distribution, samples from a bootstrap distribution, or a Bayesian posterior, you can supply samples to the `x` or `y` aesthetic.

To visualize analytical distributions, you can use the `xdist` or `ydist` aesthetic. For historical reasons, you can also use `dist` to specify the distribution, though this is not recommended as it does not work as well with orientation detection. These aesthetics can be used as follows:

- `xdist`, `ydist`, and `dist` can be any distribution object from the `distributional` package (`dist_normal()`, `dist_beta()`, etc) or can be a `posterior::rvar()` object. Since these functions are vectorized, other columns can be passed directly to them in an `aes()` specification; e.g. `aes(dist = dist_normal(mu, sigma))` will work if `mu` and `sigma` are columns in the input data frame.
- `dist` can be a character vector giving the distribution name. Then the `arg1, ... arg9` aesthetics (or `args` as a list column) specify distribution arguments. Distribution names should correspond to R functions that have "p", "q", and "d" functions; e.g. "norm" is a valid distribution name because R defines the `pnorm()`, `qnorm()`, and `dnorm()` functions for Normal distributions. See the `parse_dist()` function for a useful way to generate `dist` and `args` values from human-readable distribution specs (like "normal(0,1)"). Such specs are also produced by other packages (like the `brms::get_prior` function in brms); thus, `parse_dist()` combined with the stats described here can help you visualize the output of those functions.

Value

A `ggplot2::Geom` or `ggplot2::Stat` representing a dotplot or combined dotplot+interval geometry which can be added to a `ggplot()` object.

Aesthetics

The slab+interval stats and geoms have a wide variety of aesthetics that control the appearance of their three sub-geometries: the `slab`, the `point`, and the `interval`.

These stats support the following aesthetics:
• x: x position of the geometry (when orientation = "vertical"); or sample data to be summarized (when orientation = "horizontal" with sample data).

• y: y position of the geometry (when orientation = "horizontal"); or sample data to be summarized (when orientation = "vertical" with sample data).

• xdist: When using analytical distributions, distribution to map on the x axis: a distributional object (e.g. dist_normal()) or a posterior::rvar() object.

• ydist: When using analytical distributions, distribution to map on the y axis: a distributional object (e.g. dist_normal()) or a posterior::rvar() object.

• dist: When using analytical distributions, a name of a distribution (e.g. "norm"), a distributional object (e.g. dist_normal()), or a posterior::rvar() object. See Details.

• args: Distribution arguments (args or arg1, ... arg9). See Details.

In addition, in their default configuration (paired with geom_dotsinterval()) the following aesthetics are supported by the underlying geom:

**Slab-specific aesthetics**

• thickness: The thickness of the slab at each x value (if orientation = "horizontal") or y value (if orientation = "vertical") of the slab.

• side: Which side to place the slab on. "topright", "top", and "right" are synonyms which cause the slab to be drawn on the top or the right depending on if orientation is "horizontal" or "vertical". "bottomleft", "bottom", and "left" are synonyms which cause the slab to be drawn on the bottom or the left depending on if orientation is "horizontal" or "vertical". "topleft" causes the slab to be drawn on the top or the left, and "bottomright" causes the slab to be drawn on the bottom or the right. "both" draws the slab mirrored on both sides (as in a violin plot).

• scale: What proportion of the region allocated to this geom to use to draw the slab. If scale = 1, slabs that use the maximum range will just touch each other. Default is 0.9 to leave some space.

• justification: Justification of the interval relative to the slab, where 0 indicates bottom/left justification and 1 indicates top/right justification (depending on orientation). If justification is NULL (the default), then it is set automatically based on the value of side: when side is "top"/"right" justification is set to 0, when side is "bottom"/"left" justification is set to 1, and when side is "both" justification is set to 0.5.

• datatype: When using composite geoms directly without a stat (e.g. geom_slabinterval()), datatype is used to indicate which part of the geom a row in the data targets: rows with datatype = "slab" target the slab portion of the geometry and rows with datatype = "interval" target the interval portion of the geometry. This is set automatically when using ggdist stats.

**Binning aesthetics**

• order: The order in which data points are stacked within bins. Can be used to create the effect of "stacked" dots by ordering dots according to a discrete variable. If omitted (NULL), the value of the data points themselves are used to determine stacking order. Only applies when layout = "bin", as the other layout methods fully determine both x and y positions.

**Interval-specific aesthetics**
• xmin: Left end of the interval sub-geometry (if orientation = "horizontal").
• xmax: Right end of the interval sub-geometry (if orientation = "horizontal").
• ymin: Lower end of the interval sub-geometry (if orientation = "vertical").
• ymax: Upper end of the interval sub-geometry (if orientation = "vertical").

Point-specific aesthetics

• shape: Shape type used to draw the point sub-geometry.

Color aesthetics

• colour: (or color) The color of the interval and point sub-geometries. Use the slab_color, interval_color, or point_color aesthetics (below) to set sub-geometry colors separately.
• fill: The fill color of the slab and point sub-geometries. Use the slab_fill or point_fill aesthetics (below) to set sub-geometry colors separately.
• alpha: The opacity of the slab, interval, and point sub-geometries. Use the slab_alpha, interval_alpha, or point_alpha aesthetics (below) to set sub-geometry colors separately.
• colour_ramp: (or color_ramp) A secondary scale that modifies the color scale to "ramp" to another color. See scale_colour_ramp() for examples.
• fill_ramp: A secondary scale that modifies the fill scale to "ramp" to another color. See scale_fill_ramp() for examples.

Line aesthetics

• size: Width of the outline around the slab (if visible). Also determines the width of the line used to draw the interval and the size of the point, but raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the slab_size, interval_size, or point_size aesthetics (below) to set sub-geometry line widths separately (note that when size is set directly using the override aesthetics, interval and point sizes are not affected by interval_size_domain, interval_size_range, and fatten_point).
• linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the interval and the outline of the slab (if it is visible). Use the slab_linetype or interval_linetype aesthetics (below) to set sub-geometry line types separately.

Slab-specific color/line override aesthetics

• slab_fill: Override for fill: the fill color of the slab.
• slab_colour: (or slab_color) Override for colour/color: the outline color of the slab.
• slab_alpha: Override for alpha: the opacity of the slab.
• slab_size: Override for size: the width of the outline of the slab.
• slab_linetype: Override for linetype: the line type of the outline of the slab.
• slab_shape: Override for shape: the shape of the dots used to draw the dotplot slab.

Interval-specific color/line override aesthetics
• `interval_colour` (or `interval_color`): Override for colour/color: the color of the interval.
• `interval_alpha`: Override for alpha: the opacity of the interval.
• `interval_size`: Override for size: the line width of the interval.
• `interval_linetype`: Override for linetype: the line type of the interval.

**Point-specific color/line override aesthetics**

• `point_fill`: Override for fill: the fill color of the point.
• `point_colour` (or `point_color`): Override for colour/color: the outline color of the point.
• `point_alpha`: Override for alpha: the opacity of the point.
• `point_size`: Override for size: the size of the point.

**Other aesthetics** (these work as in standard geoms)

• `family`
• `width`
• `height`
• `group`

See examples of some of these aesthetics in action in vignette("dotsinterval"). Learn more about the sub-geom override aesthetics (like `interval_color`) in the scales documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").

**Author(s)**

Matthew Kay

**References**


**See Also**

See the `stat_slabinterval()` family for other stats built on top of `geom_slabinterval()`. See vignette("dotsinterval") for a variety of examples of use.
Examples

```r
library(dplyr)
library(ggplot2)

data(RankCorr_u_tau, package = "ggdist")

# orientation is detected automatically based on
# which axis is discrete

RankCorr_u_tau %>%
ggplot(aes(x = u_tau)) + 
  geom_dots()

RankCorr_u_tau %>%
ggplot(aes(y = u_tau)) + 
  geom_dots()

# stat_dots can summarize quantiles, creating quantile dotplots

RankCorr_u_tau %>%
ggplot(aes(x = u_tau, y = factor(i))) + 
  stat_dots(quantiles = 100)

# color and fill aesthetics can be mapped within the geom
# dotsinterval adds an interval

RankCorr_u_tau %>%
ggplot(aes(x = u_tau, y = factor(i), fill = stat(x > 6))) + 
  stat_dotsinterval(quantiles = 100)
```

---

**geom_interval**

*Multiple-interval plot (shortcut geom)*

**Description**

Shortcut version of `geom_slabinterval()` for creating multiple-interval plots.

Roughly equivalent to:

```r
geom_slabinterval(
  aes(datatype = "interval", side = "both"),
  interval_size_range = c(1, 6), show_slab = FALSE, show_point = FALSE
)
```
Usage

geom_interval(  
  mapping = NULL,  
  data = NULL,  
  stat = "identity",  
  position = "identity",  
  ...,  
  orientation = NA,  
  interval_size_range = c(1, 6),  
  interval_size_domain = c(1, 6),  
  na.rm = FALSE,  
  show.legend = NA,  
  inherit.aes = TRUE  
)

Arguments

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

data The data to be displayed in this layer. There are three options:
  If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot().
  A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.
  A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).

stat The statistical transformation to use on the data for this layer, as a string.

position Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" (position_dodge()) or "dodgejust" (position_dodgejust()) can be useful if you have overlapping geometries.

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

orientation Whether this geom is drawn horizontally or vertically. One of:
  • NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
  • "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify different groups. For each group, uses the x, xmin, xmax, and thickness aesthetics to draw points, intervals, and slabs.
  • "vertical" (or "x"): draw vertically, using the x aesthetic to identify different groups. For each group, uses the y, ymin, ymax, and thickness aesthetics to draw points, intervals, and slabs.
For compatibility with the base ggplot naming scheme for orientation, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal" (tidybayes had an orientation parameter before base ggplot did, hence the discrepancy).

**interval_size_range**
A length-2 numeric vector. This geom scales the raw size aesthetic values when drawing interval and point sizes, as they tend to be too thick when using the default settings of `scale_size_continuous()`, which give sizes with a range of c(1, 6). The `interval_size_domain` value indicates the input domain of raw size values (typically this should be equal to the value of the range argument of the `scale_size_continuous()` function), and `interval_size_range` indicates the desired output range of the size values (the min and max of the actual sizes used to draw intervals). Most of the time it is not recommended to change the value of this argument, as it may result in strange scaling of legends; this argument is a holdover from earlier versions that did not have size aesthetics targeting the point and interval separately. If you want to adjust the size of the interval or points separately, you can instead use the `interval_size` or `point_size` aesthetics; see `scales`.

**interval_size_domain**
A length-2 numeric vector giving the minimum and maximum of the values of the size aesthetic that will be translated into actual sizes for intervals drawn according to `interval_size_range` (see the documentation for that argument.)

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

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

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

### Details
This geom wraps `geom_slabinterval()` with defaults designed to produce multiple-interval plots. Default aesthetic mappings are applied if the `.width` column is present in the input data (e.g., as generated by the `point_interval()` family of functions), making this geom often more convenient than vanilla `ggplot2` geometries when used with functions like `median_qi()`, `mean_qi()`, `mode_hdi()`, etc.

Specifically, if `.width` is present in the input, `geom_interval()` acts as if its default aesthetics are `aes(colour = forcats::fct_rev(ordered(.width)))`

### Value
A `ggplot2::Geom` representing a multiple-interval geometry which can be added to a `ggplot()` object.
Aesthetics

The slab+interval stats and geoms have a wide variety of aesthetics that control the appearance of their three sub-geometries: the slab, the point, and the interval.

Positional aesthetics

• x: x position of the geometry
• y: y position of the geometry

Interval-specific aesthetics

• xmin: Left end of the interval sub-geometry (if orientation = "horizontal").
• xmax: Right end of the interval sub-geometry (if orientation = "horizontal").
• ymin: Lower end of the interval sub-geometry (if orientation = "vertical").
• ymax: Upper end of the interval sub-geometry (if orientation = "vertical").

Color aesthetics

• colour: (or color) The color of the interval and point sub-geometries. Use the slab_color, interval_color, or point_color aesthetics (below) to set sub-geometry colors separately.
• fill: The fill color of the slab and point sub-geometries. Use the slab_fill or point_fill aesthetics (below) to set sub-geometry colors separately.
• alpha: The opacity of the slab, interval, and point sub-geometries. Use the slab_alpha, interval_alpha, or point_alpha aesthetics (below) to set sub-geometry colors separately.
• colour_ramp: (or color_ramp) A secondary scale that modifies the color scale to "ramp" to another color. See scale_colour_ramp() for examples.
• fill_ramp: A secondary scale that modifies the fill scale to "ramp" to another color. See scale_fill_ramp() for examples.

Line aesthetics

• size: Width of the outline around the slab (if visible). Also determines the width of the line used to draw the interval and the size of the point, but raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the slab_size, interval_size, or point_size aesthetics (below) to set sub-geometry line widths separately (note that when size is set directly using the override aesthetics, interval and point sizes are not affected by interval_size_domain, interval_size_range, and fatten_point).
• stroke: Width of the outline around the point sub-geometry.
• linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the interval and the outline of the slab (if it is visible). Use the slab_linetype or interval_linetype aesthetics (below) to set sub-geometry line types separately.

Interval-specific color/line override aesthetics

• interval_colour: (or interval_color) Override for colour/color: the color of the interval.
- `interval_alpha`: Override for alpha: the opacity of the interval.
- `interval_size`: Override for size: the line width of the interval.
- `interval_linetype`: Override for linetype: the line type of the interval.

**Other aesthetics** (these work as in standard geoms)

- `width`
- `height`
- `group`

See examples of some of these aesthetics in action in vignette("slabinterval"). Learn more about the sub-geom override aesthetics (like `interval_color`) in the `scales` documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").

**See Also**

See `stat_interval()` for the stat version, intended for use on sample data or analytical distributions. See `geom_slabinterval()` for the geometry this shortcut is based on.

Other slabinterval geoms: `geom_pointinterval()`, `geom_slab()`

**Examples**

```r
library(dplyr)
library(ggplot2)

theme_set(theme_ggdist())

data(RankCorr_u_tau, package = "ggdist")

# orientation is detected automatically based on
# use of xmin/xmax or ymin/ymax

RankCorr_u_tau %>%
group_by(i) %>%
median_qi(.width = c(.5, .8, .95, .99)) %>%
ggplot(aes(y = i, x = u_tau, xmin = .lower, xmax = .upper)) +
geom_interval() +
scale_color_brewer()

RankCorr_u_tau %>%
group_by(i) %>%
median_qi(.width = c(.5, .8, .95, .99)) %>%
ggplot(aes(x = i, y = u_tau, ymin = .lower, ymax = .upper)) +
geom_interval() +
scale_color_brewer()
```
**Description**

A combination of `geom_line()` and `geom_ribbon()` with default aesthetics designed for use with output from `point_interval()`.

**Usage**

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

**Arguments**

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

- **data**: The data to be displayed in this layer. There are three options:
  - If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
  - A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
  - A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x, 10)`).

- **stat**: The statistical transformation to use on the data for this layer, as a string.

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

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

- **step**: Should the line/ribbon be drawn as a step function? One of:
  - FALSE (default): do not draw as a step function.
• "mid" (or TRUE): draw steps midway between adjacent x values.
• "hv": draw horizontal-then-vertical steps.
• "vh": draw as vertical-then-horizontal steps.

TRUE is an alias for "mid" because for a step function with ribbons, "mid" is probably what you want (for the other two step approaches the ribbons at either the very first or very last x value will not be visible).

**orientation**

Whether this geom is drawn horizontally or vertically. One of:

• NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
• "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify different groups. For each group, uses the x, xmin, xmax, and thickness aesthetics to draw points, intervals, and slabs.
• "vertical" (or "x"): draw vertically, using the x aesthetic to identify different groups. For each group, uses the y, ymin, ymax, and thickness aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for orientation, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal" (tidybayes had an orientation parameter before base ggplot did, hence the discrepancy).

**na.rm**

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

**show.legend**

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

**inherit.aes**

If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. borders().
• x: x position of the geometry
• y: y position of the geometry

**Ribbon-specific aesthetics**

• xmin: Left edge of the ribbon sub-geometry (if orientation = "horizontal").
• xmax: Right edge of the ribbon sub-geometry (if orientation = "horizontal").
• ymin: Lower edge of the ribbon sub-geometry (if orientation = "vertical").
• ymax: Upper edge of the ribbon sub-geometry (if orientation = "vertical").

**Color aesthetics**

• colour: (or color) The color of the line sub-geometry.
• fill: The fill color of the ribbon sub-geometry.
• alpha: The opacity of the line and ribbon sub-geometries.
• fill_ramp: A secondary scale that modifies the fill scale to "ramp" to another color. See `scale_fill_ramp()` for examples.

**Line aesthetics**

• size: Width of line.
• linetype: Type of line (e.g., "solid", "dashed", etc)

**Other aesthetics** (these work as in standard geoms)

• group

See examples of some of these aesthetics in action in vignette("lineribbon"). Learn more about the sub-geom override aesthetics (like interval_color) in the scales documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").

**Author(s)**

Matthew Kay

**See Also**

See `stat_lineribbon()` for a version that does summarizing of samples into points and intervals within ggplot. See `geom_pointinterval()` for a similar geom intended for point summaries and intervals. See `geom_ribbon()` and `geom_line()` for the geoms this is based on.
Examples

```r
library(dplyr)
library(ggplot2)

theme_set(theme_ggdist())

tibble(x = 1:10) %>%
  group_by_all() %>%
do(tibble(y = rnorm(100, .x))) %>%
  median_qi(.width = c(.5, .8, .95)) %>%
ggplot(aes(x = x, y = y, ymin = .lower, ymax = .upper)) +
  # automatically uses aes(fill = forcats::fct_rev(ordered(.width)))
  geom_lineribbon() +
  scale_fill_brewer()
```

geom_pointinterval  
**Point + multiple-interval plot (shortcut geom)**

Description

Shortcut version of `geom_slabinterval()` for creating point + multiple-interval plots.

Roughly equivalent to:

```r
geom_slabinterval(
  aes(datatype = "interval", side = "both"),
  show_slab = FALSE,
  show.legend = c(size = FALSE)
)
```

Usage

```r
geom_pointinterval(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  orientation = NA,
  interval_size_domain = c(1, 6),
  interval_size_range = c(0.6, 1.4),
  fatten_point = 1.8,
  na.rm = FALSE,
  show.legend = c(size = FALSE),
  inherit.aes = TRUE
)
```
Arguments

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

- **data**: The data to be displayed in this layer. There are three options:
  - If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
  - A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
  - A function will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x, 10)`).

- **stat**: The statistical transformation to use on the data for this layer, as a string.

- **position**: Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" (`position_dodge()`) or "dodgejust" (`position_dodgejust()`) can be useful if you have overlapping geometries.

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

- **orientation**: Whether this geom is drawn horizontally or vertically. One of:
  - NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
  - "horizontal" (or "y"): draw horizontally, using the `y` aesthetic to identify different groups. For each group, uses the `x`, `xmin`, `xmax`, and `thickness` aesthetics to draw points, intervals, and slabs.
  - "vertical" (or "x"): draw vertically, using the `x` aesthetic to identify different groups. For each group, uses the `y`, `ymin`, `ymax`, and `thickness` aesthetics to draw points, intervals, and slabs.

  For compatibility with the base ggplot naming scheme for `orientation`, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal" (tidybayes had an `orientation` parameter before base ggplot did, hence the discrepancy).

- **interval_size_domain**: A length-2 numeric vector giving the minimum and maximum of the values of the `size` aesthetic that will be translated into actual sizes for intervals drawn according to `interval_size_range` (see the documentation for that argument).

- **interval_size_range**: A length-2 numeric vector. This geom scales the raw `size` aesthetic values when drawing interval and point sizes, as they tend to be too thick when using the default settings of `scale_size_continuous()`, which give sizes with a range of c(1, 6). The `interval_size_domain` value indicates the input domain of raw `size` values (typically this should be equal to the value of the `range` argument of the `scale_size_continuous()` function), and `interval_size_range` indicates the desired output range of the size values (the min and max of the...
geom_pointinterval

actual sizes used to draw intervals). Most of the time it is not recommended to change the value of this argument, as it may result in strange scaling of legends; this argument is a holdover from earlier versions that did not have size aesthetics targeting the point and interval separately. If you want to adjust the size of the interval or points separately, you can instead use the interval_size or point_size aesthetics; see scales.

fatten_point
A multiplicative factor used to adjust the size of the point relative to the size of the thickest interval line. If you wish to specify point sizes directly, you can also use the point_size aesthetic and scale_point_size_continuous() or scale_point_size_discrete(); sizes specified with that aesthetic will not be adjusted using fatten_point.

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

show.legend
Should this layer be included in the legends? Default is c(size = FALSE), unlike most geoms, to match its common use cases. FALSE hides all legends, TRUE shows all legends, and NA shows only those that are mapped (the default for most geoms).

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

Details
This geom wraps geom_slabinterval() with defaults designed to produce point + multiple-interval plots. Default aesthetic mappings are applied if the .width column is present in the input data (e.g., as generated by the point_interval() family of functions), making this geom often more convenient than vanilla ggplot2 geometries when used with functions like median_qi(), mean_qi(), mode_hdi(), etc.

Specifically, if .width is present in the input, geom_pointinterval() acts as if its default aesthetics areaes(size = -.width)

Value
A ggplot2::Geom representing a point + multiple-interval geometry which can be added to a ggplot() object.

Aesthetics
The slab+interval stats and geoms have a wide variety of aesthetics that control the appearance of their three sub-geometries: the slab, the point, and the interval.

Positional aesthetics
- x: x position of the geometry
- y: y position of the geometry

Interval-specific aesthetics
- xmin: Left end of the interval sub-geometry (if orientation = "horizontal").
• `xmax`: Right end of the interval sub-geometry (if orientation = "horizontal").
• `ymin`: Lower end of the interval sub-geometry (if orientation = "vertical").
• `ymax`: Upper end of the interval sub-geometry (if orientation = "vertical").

Point-specific aesthetics

• `shape`: Shape type used to draw the `point` sub-geometry.

Color aesthetics

• `colour` (or `color`): The color of the `interval` and `point` sub-geometries. Use the `slab_color`, `interval_color`, or `point_color` aesthetics (below) to set sub-geometry colors separately.

• `fill`: The fill color of the `slab` and `point` sub-geometries. Use the `slab_fill` or `point_fill` aesthetics (below) to set sub-geometry colors separately.

• `alpha`: The opacity of the `slab`, `interval`, and `point` sub-geometries. Use the `slab_alpha`, `interval_alpha`, or `point_alpha` aesthetics (below) to set sub-geometry colors separately.

• `colour_ramp` (or `color_ramp`): A secondary scale that modifies the color scale to "ramp" to another color. See `scale_colour_ramp()` for examples.

• `fill_ramp`: A secondary scale that modifies the fill scale to "ramp" to another color. See `scale_fill_ramp()` for examples.

Line aesthetics

• `size`: Width of the outline around the `slab` (if visible). Also determines the width of the line used to draw the `interval` and the size of the `point`, but raw size values are transformed according to the `interval_size_domain`, `interval_size_range`, and `fatten_point` parameters of the `geom` (see above). Use the `slab_size`, `interval_size`, or `point_size` aesthetics (below) to set sub-geometry line widths separately (note that when size is set directly using the override aesthetics, interval and point sizes are not affected by `interval_size_domain`, `interval_size_range`, and `fatten_point`).

• `stroke`: Width of the outline around the `point` sub-geometry.

• `linetype`: Type of line (e.g., "solid", "dashed", etc) used to draw the `interval` and the outline of the `slab` (if it is visible). Use the `slab_linetype` or `interval_linetype` aesthetics (below) to set sub-geometry line types separately.

Interval-specific color/line override aesthetics

• `interval_colour` (or `interval_color`): Override for `colour/color`: the color of the interval.

• `interval_alpha`: Override for `alpha`: the opacity of the interval.

• `interval_size`: Override for `size`: the line width of the interval.

• `interval_linetype`: Override for `linetype`: the line type of the interval.

Point-specific color/line override aesthetics

• `point_fill`: Override for `fill`: the fill color of the point.

• `point_colour` (or `point_color`): Override for `colour/color`: the outline color of the point.
geom_pointinterval

- point_alpha: Override for alpha: the opacity of the point.
- point_size: Override for size: the size of the point.

Other aesthetics (these work as in standard geoms)

- width
- height
- group

See examples of some of these aesthetics in action in vignette("slabinterval"). Learn more about the sub-geom override aesthetics (like interval_color) in the scales documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").

See Also

See stat_pointinterval() for the stat version, intended for use on sample data or analytical distributions. See geom_slabinterval() for the geometry this shortcut is based on.

Other slabinterval geoms: geom_interval(), geom_slab()

Examples

library(dplyr)
library(ggplot2)

data(RankCorr_u_tau, package = "ggdist")

# orientation is detected automatically based on
# use of xmin/xmax or ymin/ymax

RankCorr_u_tau %>%
group_by(i) %>%
median_qi(.width = c(.8, .95)) %>%
ggplot(aes(y = i, x = u_tau, xmin = .lower, xmax = .upper)) +
geom_pointinterval()

RankCorr_u_tau %>%
group_by(i) %>%
median_qi(.width = c(.8, .95)) %>%
ggplot(aes(x = i, y = u_tau, ymin = .lower, ymax = .upper)) +
geom_pointinterval()
Description

Shortcut version of `geom_slabinterval()` for creating slab (ridge) plots.
Roughly equivalent to:

```r
geom_slabinterval(
  show_point = FALSE, show_interval = FALSE
)
```

Usage

```r
geom_slab(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ..., 
  orientation = NA,
  normalize = "all",
  fill_type = "segments",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

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

**orientation**

Whether this geom is drawn horizontally or vertically. One of:

- `NA` (default): automatically detect the orientation based on how the aesthet-
icities are assigned. Automatic detection works most of the time.
- "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify
different groups. For each group, uses the x, xmin, xmax, and thickness
aesthetics to draw points, intervals, and slabs.
- "vertical" (or "x"): draw vertically, using the x aesthetic to identify dif-
ferent groups. For each group, uses the y, ymin, ymax, and thickness
aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for orientation, "x"
can be used as an alias for "vertical" and "y" as an alias for "horizontal"
tidybayes had an orientation parameter before base ggplot did, hence the
discrepancy).

**normalize**

How to normalize heights of functions input to the thickness aesthetic. One
of:

- "all": normalize so that the maximum height across all data is 1.
- "panels": normalize within panels so that the maximum height in each
panel is 1.
- "xy": normalize within the x/y axis opposite the orientation of this geom
so that the maximum height at each value of the opposite axis is 1.
- "groups": normalize within values of the opposite axis and within each
group so that the maximum height in each group is 1.
- "none": values are taken as is with no normalization (this should probably
only be used with functions whose values are in [0,1], such as CDFs).

**fill_type**

What type of fill to use when the fill color or alpha varies within a slab. One of:

- "segments": breaks up the slab geometry into segments for each unique
combination of fill color and alpha value. This approach is supported by all
graphics devices and works well for sharp cutoff values, but can give ugly
results if a large number of unique fill colors are being used (as in gradients,
like in `stat_gradientinterval()`).
- "gradient": a `grid::linearGradient()` is used to create a smooth gra-
dient fill. This works well for large numbers of unique fill colors, but re-
quires R >= 4.1 and is not yet supported on all graphics devices. As of
this writing, the `png()` graphics device with `type = "cairo"`, the `svg()`
device, the `pdf()` device, and the `ragg::agg_png()` devices are known to
support this option. On R < 4.1, this option will fall back to `fill_type = "segment"` with a message.
- "auto": attempts to use `fill_type = "gradient"` if support for it can be
auto-detected. On R >= 4.2, support for gradients can be auto-detected
on some graphics devices; if support is not detected, this option will fall
back to `fill_type = "segments"` (in case of a false negative, `fill_type = "gradient"` can be set explicitly). On R < 4.2, support for gradients cannot
be auto-detected, so this will always fall back to fill_type = "segments", in which case you can set fill_type = "gradient" explicitly if you are using a graphics device that supports gradients.

- **na.rm** If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
- **show.legend** logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
- **inherit.aes** If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. borders().

**Value**

A ggplot2::Geom representing a slab (ridge) geometry which can be added to a ggplot() object.

**Aesthetics**

The slab+interval stats and geoms have a wide variety of aesthetics that control the appearance of their three sub-geometries: the slab, the point, and the interval.

**Positional aesthetics**

- **x**: x position of the geometry
- **y**: y position of the geometry

**Slab-specific aesthetics**

- **thickness**: The thickness of the slab at each x value (if orientation = "horizontal") or y value (if orientation = "vertical") of the slab.
- **side**: Which side to place the slab on. "topright", "top", and "right" are synonyms which cause the slab to be drawn on the top or the right depending on if orientation is "horizontal" or "vertical". "bottomleft", "bottom", and "left" are synonyms which cause the slab to be drawn on the bottom or the left depending on if orientation is "horizontal" or "vertical". "topleft" causes the slab to be drawn on the top or the left, and "bottomright" causes the slab to be drawn on the bottom or the right. "both" draws the slab mirrored on both sides (as in a violin plot).
- **scale**: What proportion of the region allocated to this geom to use to draw the slab. If scale = 1, slabs that use the maximum range will just touch each other. Default is 0.9 to leave some space.
- **justification**: Justification of the interval relative to the slab, where 0 indicates bottom/left justification and 1 indicates top/right justification (depending on orientation). If justification is NULL (the default), then it is set automatically based on the value of side: when side is "top"/"right" justification is set to 0, when side is "bottom"/"left" justification is set to 1, and when side is "both" justification is set to 0.5.
- **datatype**: When using composite geoms directly without a stat (e.g. geom_slabinterval()), datatype is used to indicate which part of the geom a row in the data targets: rows with datatype = "slab" target the slab portion of the geometry and rows with datatype = "interval" target the interval portion of the geometry. This is set automatically when using ggdist stats.
geom_slab

Color aesthetics

- colour: (or color) The color of the interval and point sub-geometries. Use the slab_color, interval_color, or point_color aesthetics (below) to set sub-geometry colors separately.
- fill: The fill color of the slab and point sub-geometries. Use the slab_fill or point_fill aesthetics (below) to set sub-geometry colors separately.
- alpha: The opacity of the slab, interval, and point sub-geometries. Use the slab_alpha, interval_alpha, or point_alpha aesthetics (below) to set sub-geometry colors separately.
- colour_ramp: (or color_ramp) A secondary scale that modifies the color scale to "ramp" to another color. See scale_colour_ramp() for examples.
- fill_ramp: A secondary scale that modifies the fill scale to "ramp" to another color. See scale_fill_ramp() for examples.

Line aesthetics

- size: Width of the outline around the slab (if visible). Also determines the width of the line used to draw the interval and the size of the point, but raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the slab_size, interval_size, or point_size aesthetics (below) to set sub-geometry line widths separately (note that when size is set directly using the override aesthetics, interval and point sizes are not affected by interval_size_domain, interval_size_range, and fatten_point).
- stroke: Width of the outline around the point sub-geometry.
- linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the interval and the outline of the slab (if it is visible). Use the slab_linetype or interval_linetype aesthetics (below) to set sub-geometry line types separately.

Slab-specific color/line override aesthetics

- slab_fill: Override for fill: the fill color of the slab.
- slab_colour: (or slab_color) Override for colour/color: the outline color of the slab.
- slab_alpha: Override for alpha: the opacity of the slab.
- slab_size: Override for size: the width of the outline of the slab.
- slab_linetype: Override for linetype: the line type of the outline of the slab.

Other aesthetics (these work as in standard geoms)

- width
- height
- group

See examples of some of these aesthetics in action in vignette("slabinterval"). Learn more about the sub-geom override aesthetics (like interval_color) in the scales documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").
See Also

See `stat_slab()` for the stat version, intended for use on sample data or analytical distributions. See `geom_slabinterval()` for the geometry this shortcut is based on.

Other slabinterval geoms: `geom_interval()`, `geom_pointinterval()`

Examples

```r
library(dplyr)
library(ggplot2)

theme_set(theme_ggdist())

# we will manually demonstrate plotting a density with geom_slab(),
# though generally speaking this is easier to do using stat_slab(), which
# will determine sensible limits automatically and correctly adjust
# densities when using scale transformations
df = expand.grid(
  mean = 1:3,
  input = seq(-2, 6, length.out = 100)
) %>%
mutate(
  group = letters[4 - mean],
  density = dnorm(input, mean, 1)
)

# orientation is detected automatically based on
# use of x or y
df %>%
ggplot(aes(y = group, x = input, thickness = density)) +
  geom_slab()

df %>%
ggplot(aes(x = group, y = input, thickness = density)) +
  geom_slab()

# RIDGE PLOTS
# "ridge" plots can be created by increasing the slab height and
# setting the slab color
df %>%
ggplot(aes(y = group, x = input, thickness = density)) +
  geom_slab(height = 2, color = "black")
```

---

`geom_slabinterval`  
*Slab + point + interval meta-geom*
Description

This meta-geom supports drawing combinations of functions (as slabs, aka ridge plots or joy plots), points, and intervals. It acts as a meta-geom for many other tidybayes geoms that are wrappers around this geom, including eye plots, half-eye plots, CCDF barplots, and point+multiple interval plots, and supports both horizontal and vertical orientations, dodging (via the position argument), and relative justification of slabs with their corresponding intervals.

Usage

```r
geom_slabinterval(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ..., 
  orientation = NA,
  normalize = "all",
  fill_type = "segments",
  interval_size_domain = c(1, 6),
  interval_size_range = c(0.6, 1.4),
  fatten_point = 1.8,
  show_slab = TRUE,
  show_point = TRUE,
  show_interval = TRUE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

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

- **data**: The data to be displayed in this layer. There are three options:
  - If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
  - A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
  - A function will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x, 10)`).

- **stat**: The statistical transformation to use on the data for this layer, as a string.

- **position**: Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" (`position_dodge()`) or "dodgejust" (`position_dodgejust()`) can be useful if you have overlapping geometries.
Other arguments passed to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3` (see Aesthetics, below). They may also be parameters to the paired geom/stat.

**orientation** Whether this geom is drawn horizontally or vertically. One of:
- NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
- "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify different groups. For each group, uses the x, xmin, xmax, and thickness aesthetics to draw points, intervals, and slabs.
- "vertical" (or "x"): draw vertically, using the x aesthetic to identify different groups. For each group, uses the y, ymin, ymax, and thickness aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for orientation, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal" (tidybayes had an orientation parameter before base ggplot did, hence the discrepancy).

**normalize** How to normalize heights of functions input to the thickness aesthetic. One of:
- "all": normalize so that the maximum height across all data is 1.
- "panels": normalize within panels so that the maximum height in each panel is 1.
- "xy": normalize within the x/y axis opposite the orientation of this geom so that the maximum height at each value of the opposite axis is 1.
- "groups": normalize within values of the opposite axis and within each group so that the maximum height in each group is 1.
- "none": values are taken as is with no normalization (this should probably only be used with functions whose values are in [0,1], such as CDFs).

**fill_type** What type of fill to use when the fill color or alpha varies within a slab. One of:
- "segments": breaks up the slab geometry into segments for each unique combination of fill color and alpha value. This approach is supported by all graphics devices and works well for sharp cutoff values, but can give ugly results if a large number of unique fill colors are being used (as in gradients, like in `stat_gradientinterval()`).
- "gradient": a `grid::linearGradient()` is used to create a smooth gradient fill. This works well for large numbers of unique fill colors, but requires R >= 4.1 and is not yet supported on all graphics devices. As of this writing, the `png()` graphics device with type = "cairo", the `svg()` device, the `pdf()` device, and the `ragg::agg_png()` devices are known to support this option. On R < 4.1, this option will fall back to `fill_type = "segment"` with a message.
- "auto": attempts to use `fill_type = "gradient"` if support for it can be auto-detected. On R >= 4.2, support for gradients can be auto-detected on some graphics devices; if support is not detected, this option will fall back to `fill_type = "segments"` (in case of a false negative, `fill_type = "gradient"` can be set explicitly). On R < 4.2, support for gradients cannot
be auto-detected, so this will always fall back to `fill_type = "segments"`, in which case you can set `fill_type = "gradient"` explicitly if you are using a graphics device that support gradients.

### interval_size_domain
A length-2 numeric vector giving the minimum and maximum of the values of the size aesthetic that will be translated into actual sizes for intervals drawn according to `interval_size_range` (see the documentation for that argument.)

### interval_size_range
A length-2 numeric vector. This geom scales the raw size aesthetic values when drawing interval and point sizes, as they tend to be too thick when using the default settings of `scale_size_continuous()`, which give sizes with a range of c(1, 6). The `interval_size_domain` value indicates the input domain of raw size values (typically this should be equal to the value of the range argument of the `scale_size_continuous()` function), and `interval_size_range` indicates the desired output range of the size values (the min and max of the actual sizes used to draw intervals). Most of the time it is not recommended to change the value of this argument, as it may result in strange scaling of legends; this argument is a holdover from earlier versions that did not have size aesthetics targeting the point and interval separately. If you want to adjust the size of the interval or points separately, you can instead use the `interval_size` or `point_size` aesthetics; see `scales`.

### fatten_point
A multiplicative factor used to adjust the size of the point relative to the size of the thickest interval line. If you wish to specify point sizes directly, you can also use the `point_size` aesthetic and `scale_point_size_continuous()` or `scale_point_size_discrete()`: sizes specified with that aesthetic will not be adjusted using `fatten_point`.

### show_slab
Should the slab portion of the geom be drawn?

### show_point
Should the point portion of the geom be drawn?

### show_interval
Should the interval portion of the geom be drawn?

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

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

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

## Details

`geom_slabinterval()` is a flexible meta-geom that you can use directly or through a variety of "shortcut" geoms that represent useful combinations of the various parameters of this geom. In many cases you will want to use the shortcut geoms instead as they create more useful mnemonic primitives, such as eye plots, half-eye plots, point-interval plots, or CCDF barplots.

The slab portion of the geom is much like a ridge or "joy" plot: it represents the value of a function scaled to fit between values on the x or y axis (depending on the value of orientation). Values of
the functions are specified using the thickness aesthetic and are scaled to fit into scale times the distance between points on the relevant axis. E.g., if orientation is "horizontal", scale is 0.9, and y is a discrete variable, then the thickness aesthetic specifies the value of some function of x that is drawn for every y value and scaled to fit into 0.9 times the distance between points on the y axis.

For the interval portion of the geom, x and y aesthetics specify the location of the point, and ymin/ymax or xmin/xmax (depending on the value of orientation) specify the endpoints of the interval. A scaling factor for interval line width and point size is applied through the interval_size_domain, interval_size_range, and fatten_point parameters. These scaling factors are designed to give multiple uncertainty intervals reasonable scaling at the default settings for scale_size_continuous().

As a combination geom, this geom expects a datatype aesthetic specifying which part of the geom a given row in the input data corresponds to: "slab" or "interval". However, specifying this aesthetic manually is typically only necessary if you use this geom directly; the numerous wrapper geoms will usually set this aesthetic for you as needed, and their use is recommended unless you have a very custom use case.

Wrapper geoms include:
- geom_pointinterval()
- geom_interval()
- geom_slab()

In addition, the stat_slabinterval() family of stats uses geoms from the geom_slabinterval() family, and is often easier to use than using these geoms directly. Typically, the geom_* versions are meant for use with already-summarized data (such as intervals) and the stat_* versions are summarize the data themselves (usually draws from a distribution) to produce the geom.

Value

A ggplot2::Geom representing a slab or combined slab+interval geometry which can be added to a ggplot() object.

Aesthetics

The slab+interval stats and geoms have a wide variety of aesthetics that control the appearance of their three sub-geometries: the slab, the point, and the interval.

Positional aesthetics

- x: x position of the geometry
- y: y position of the geometry

Slab-specific aesthetics

- thickness: The thickness of the slab at each x value (if orientation = "horizontal") or y value (if orientation = "vertical") of the slab.
- side: Which side to place the slab on. "topright", "top", and "right" are synonyms which cause the slab to be drawn on the top or the right depending on if orientation is "horizontal" or "vertical". "bottomleft", "bottom", and "left" are synonyms which cause the slab to be drawn on the bottom or the left depending on if orientation
is "horizontal" or "vertical". "topleft" causes the slab to be drawn on the top or the left, and "bottomright" causes the slab to be drawn on the bottom or the right. "both" draws the slab mirrored on both sides (as in a violin plot).

- **scale**: What proportion of the region allocated to this geom to use to draw the slab. If scale = 1, slabs that use the maximum range will just touch each other. Default is 0.9 to leave some space.

- **justification**: Justification of the interval relative to the slab, where 0 indicates bottom/left justification and 1 indicates top/right justification (depending on orientation). If justification is NULL (the default), then it is set automatically based on the value of side: when side is "top"/"right" justification is set to 0, when side is "bottom"/"left" justification is set to 1, and when side is "both" justification is set to 0.5.

- **datatype**: When using composite geoms directly without a stat (e.g. `geom_slabinterval()`), datatype is used to indicate which part of the geom a row in the data targets: rows with datatype = "slab" target the slab portion of the geometry and rows with datatype = "interval" target the interval portion of the geometry. This is set automatically when using ggdist stats.

**Interval-specific aesthetics**

- **xmin**: Left end of the interval sub-geometry (if orientation = "horizontal").
- **xmax**: Right end of the interval sub-geometry (if orientation = "horizontal").
- **ymin**: Lower end of the interval sub-geometry (if orientation = "vertical").
- **ymax**: Upper end of the interval sub-geometry (if orientation = "vertical").

**Point-specific aesthetics**

- **shape**: Shape type used to draw the point sub-geometry.

**Color aesthetics**

- **colour**: (or color) The color of the interval and point sub-geometries. Use the slab_color, interval_color, or point_color aesthetics (below) to set sub-geometry colors separately.
- **fill**: The fill color of the slab and point sub-geometries. Use the slab_fill or point_fill aesthetics (below) to set sub-geometry colors separately.
- **alpha**: The opacity of the slab, interval, and point sub-geometries. Use the slab_alpha, interval_alpha, or point_alpha aesthetics (below) to set sub-geometry colors separately.
- **colour_ramp**: (or color_ramp) A secondary scale that modifies the color scale to "ramp" to another color. See `scale_colour_ramp()` for examples.
- **fill_ramp**: A secondary scale that modifies the fill scale to "ramp" to another color. See `scale_fill_ramp()` for examples.

**Line aesthetics**

- **size**: Width of the outline around the slab (if visible). Also determines the width of the line used to draw the interval and the size of the point, but raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the slab_size, interval_size, or point_size aesthetics (below) to set sub-geometry line widths separately (note that when size is set directly using the override aesthetics, interval and point sizes are not affected by interval_size_domain, interval_size_range, and fatten_point).
• stroke: Width of the outline around the **point** sub-geometry.
• linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the **slab_linetype** or **interval_linetype** aesthetics (below) to set sub-geometry line types separately.

**Slab-specific color/line override aesthetics**

• **slab_fill**: Override for **fill**: the fill color of the slab.
• **slab_colour** (or **slab_color**) Override for **colour/color**: the outline color of the slab.
• **slab_alpha**: Override for **alpha**: the opacity of the slab.
• **slab_size**: Override for **size**: the width of the outline of the slab.
• **slab_linetype**: Override for **linetype**: the line type of the outline of the slab.

**Interval-specific color/line override aesthetics**

• **interval_colour** (or **interval_color**) Override for **colour/color**: the color of the interval.
• **interval_alpha**: Override for **alpha**: the opacity of the interval.
• **interval_size**: Override for **size**: the line width of the interval.
• **interval_linetype**: Override for **linetype**: the line type of the interval.

**Point-specific color/line override aesthetics**

• **point_fill**: Override for **fill**: the fill color of the point.
• **point_colour** (or **point_color**) Override for **colour/color**: the outline color of the point.
• **point_alpha**: Override for **alpha**: the opacity of the point.
• **point_size**: Override for **size**: the size of the point.

**Other aesthetics** (these work as in standard geoms)

• **width**
• **height**
• **group**

See examples of some of these aesthetics in action in vignette("slabinterval"). Learn more about the sub-geom override aesthetics (like **interval_color**) in the **scales** documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").

**Author(s)**

Matthew Kay

**See Also**

See **geom_lineribbon()** for a combination geom designed for fit curves plus probability bands. See **geom_dotsinterval()** for a combination geom designed for plotting dotplots with intervals. See **stat_slabinterval()** for families of stats built on top of this geom for common use cases (like **stat_halfeye()**). See vignette("slabinterval") for a variety of examples of use.
Examples

# geom_slabinterval() is typically not that useful on its own.
# See vignette("slabinterval") for a variety of examples of the use of its
# shortcut geoms and stats, which are more useful than using
# geom_slabinterval() directly.

Description

Deprecation functions and arguments and their alternatives are listed below.

Deprecated stats and geoms

The `stat_sample_...` and `stat_dist_...` families of stats were merged in ggdist 3.1. This means:

- `stat_dist_...` is deprecated. For any code using `stat_dist_XXX()`, you should now be able to use `stat_XXX()` instead without additional modifications in almost all cases.
- `stat_sample_slabinterval()` is deprecated. You should be able to use `stat_slabinterval()` instead without additional modifications in almost all cases.

The old `stat_dist_...` names are currently kept as aliases, but may be removed in the future.

Deprecated arguments

Parameters for `stat_slabinterval()` and family deprecated as of ggdist 3.1 are:

- The `.prob` argument, which is a long-deprecated alias for `.width`, was removed in ggdist 3.1.
- The `limits_function` argument: this was a parameter for determining the function to compute limits of the slab in `stat_slabinterval()` and its derived stats. This function is really an internal function only needed by subclasses of the base class, yet added a lot of noise to the documentation, so it was replaced with `AbstractStatSlabInterval$compute_limits()`.
- The `limits_args` argument: extra stat parameters are now passed through to the ... arguments to `AbstractStatSlabInterval$compute_limits()`; use these instead.
- The `slab_function` argument: this was a parameter for determining the function to compute slabs in `stat_slabinterval()` and its derived stats. This function is really an internal function only needed by subclasses of the base class, yet added a lot of noise to the documentation, so it was replaced with `AbstractStatSlabInterval$compute_slab()`.
- The `slab_args` argument: extra stat parameters are now passed through to the ... arguments to `AbstractStatSlabInterval$compute_slab()`; use these instead.
• The interval_function and fun.data arguments: these were parameters for determining
the function to compute intervals in stat_slabinterval() and its derived stats. This function
is really an internal function only needed by subclasses of the base class, yet added a lot of
noise to the documentation, so it was replaced with AbstractStatSlabInterval$compute_interval().
• The interval_args and fun.args arguments: to pass extra arguments to a point_interval
replace the value of the point_interval argument with a simple wrapper; e.g. stat_halfeye(point_interval = \
(.
Parameters for geom_slabinterval() and family deprecated as of ggdist 3.1 are:
• The size_domain and size_range arguments, which are long-deprecated aliases for interval_size_domain
and interval_size_range, were removed in ggdist 3.1.

Author(s)
Matthew Kay

---

guide_rampbar

Continuous colour ramp guide

Description
A colour ramp bar guide that shows continuous colour ramp scales mapped onto values as a smooth
gradient. Designed for use with scale_fill_ramp_continuous() and scale_colour_ramp_continuous().
Based on guide_colourbar().

Usage
guide_rampbar(
  ..., 
  to = "gray65",
  available_aes = c("fill_ramp", "colour_ramp")
)

Arguments
... Arguments passed on to ggplot2::guide_colourbar
title A character string or expression indicating a title of guide. If NULL, the
title is not shown. By default (waiver()), the name of the scale object or
the name specified in labs() is used for the title.
title.position A character string indicating the position of a title. One of
"top" (default for a vertical guide), "bottom", "left" (default for a horizontal
guide), or "right."
title.theme A theme object for rendering the title text. Usually the object
of element_text() is expected. By default, the theme is specified by
legend.title in theme() or theme.
title.hjust A number specifying horizontal justification of the title text.
title.vjust A number specifying vertical justification of the title text.
label logical. If TRUE then the labels are drawn. If FALSE then the labels are invisible.

label.position A character string indicating the position of a label. One of "top", "bottom" (default for horizontal guide), "left", or "right" (default for vertical guide).

label.theme A theme object for rendering the label text. Usually the object of \code{element_text()} is expected. By default, the theme is specified by \code{legend.text} in \code{theme()}. 

label.hjust A numeric specifying horizontal justification of the label text.

label.vjust A numeric specifying vertical justification of the label text.

barwidth A numeric or a \code{grid::unit()} object specifying the width of the colourbar. Default value is \code{legend.key.width} or \code{legend.key.size} in \code{theme()} or \code{theme()}. 

barheight A numeric or a \code{grid::unit()} object specifying the height of the colourbar. Default value is \code{legend.key.height} or \code{legend.key.size} in \code{theme()} or \code{theme()}. 

nbin A numeric specifying the number of bins for drawing the colourbar. A smoother colourbar results from a larger value.

raster A logical. If TRUE then the colourbar is rendered as a raster object. If FALSE then the colourbar is rendered as a set of rectangles. Note that not all graphics devices are capable of rendering raster image.

frame.colour A string specifying the colour of the frame drawn around the bar. If NULL (the default), no frame is drawn.

frame.linewidth A numeric specifying the width of the frame drawn around the bar.

frame.linetype A numeric specifying the linetype of the frame drawn around the bar.

ticks A logical specifying if tick marks on the colourbar should be visible.

ticks.colour A string specifying the colour of the tick marks.

ticks.linewidth A numeric specifying the width of the tick marks.

draw.ulim A logical specifying if the upper limit tick marks should be visible.

draw.llim A logical specifying if the lower limit tick marks should be visible.

direction A character string indicating the direction of the guide. One of "horizontal" or "vertical."

default.unit A character string indicating \code{grid::unit()} for \code{barwidth} and \code{barheight}.

reverse logical. If TRUE the colourbar is reversed. By default, the highest value is on the top and the lowest value is on the bottom.

order positive integer less than 99 that specifies the order of this guide among multiple guides. This controls the order in which multiple guides are displayed, not the contents of the guide itself. If 0 (default), the order is determined by a secret algorithm.
Details

This guide creates smooth gradient color bars for use with `scale_fill_ramp_continuous()` and `scale_colour_ramp_continuous()`. The color to ramp from is determined by the from argument of the scale_* function, and the color to ramp to is determined by the to argument to `guide_rampbar()`.

Guides can be specified in each scale_* function or in guides(). `guide = "rampbar"` in scale_* is syntactic sugar for `guide = guide_rampbar();` e.g. `scale_colour_ramp_continuous(guide = "rampbar")`. For how to specify the guide for each scale in more detail, see guides().

Value

A guide object.

Author(s)

Matthew Kay

See Also

`scale_fill_ramp_continuous()`, `scale_colour_ramp_continuous()`.

Examples

```r
library(dplyr)
library(ggplot2)
library(distributional)

# The default guide for ramp scales is guide_legend(), which creates a
# discrete style scale:
tibble(d = dist_uniform(0, 1)) %>%
  ggplot(aes(y = 0, xdist = d)) +
  stat_slab(aes(fill_ramp = stat(x)), fill = "blue") +
  scale_fill_ramp_continuous(from = "red")

# We can guide_rampbar() to instead create a continuous guide, but
# it does not know what color to ramp to (defaults to "gray65"):
tibble(d = dist_uniform(0, 1)) %>%
  ggplot(aes(y = 0, xdist = d)) +
  stat_slab(aes(fill_ramp = stat(x)), fill = "blue") +
  scale_fill_ramp_continuous(from = "red", guide = guide_rampbar())

# We can tell the guide what color to ramp to using the 'to' argument:
tibble(d = dist_uniform(0, 1)) %>%
  ggplot(aes(y = 0, xdist = d)) +
  stat_slab(aes(fill_ramp = stat(x)), fill = "blue") +
  scale_fill_ramp_continuous(from = "red", guide = guide_rampbar(to = "blue"))
```
Description
Marginal distribution for the correlation in a single cell from a correlation matrix distributed according to an LKJ distribution.

Usage
\[
dlkjcorr\_marginal(x, K, \eta, \log)\]
\[
plkjcorr\_marginal(q, K, \eta, \text{lower.tail}, \log.p)\]
\[
qlkjcorr\_marginal(p, K, \eta, \text{lower.tail}, \log.p)\]
\[
rlkjcorr\_marginal(n, K, \eta)\]

Arguments
- \(x, q\): vector of quantiles.
- \(K\): Dimension of the correlation matrix. Must be greater than or equal to 2.
- \(\eta\): Parameter controlling the shape of the distribution
- \(\log, \log.p\): logical; if TRUE, probabilities \(p\) are given as \(\log(p)\).
- \(\text{lower.tail}\): logical; if TRUE (default), probabilities are \(P[X \leq x]\) otherwise, \(P[X > x]\).
- \(p\): vector of probabilities.
- \(n\): number of observations. If \(\text{length}(n) > 1\), the length is taken to be the number required.

Details
The LKJ distribution is a distribution over correlation matrices with a single parameter, \(\eta\). For a given \(\eta\) and a \(K \times K\) correlation matrix \(R\):
\[
R \sim \text{LKJ}(\eta)
\]
Each off-diagonal entry of \(R\), \(r_{ij} : i \neq j\), has the following marginal distribution (Lewandowski, Kurowicka, and Joe 2009):
\[
\frac{r_{ij} + 1}{2} \sim \text{Beta}\left(\eta - 1 + \frac{K}{2}, \eta - 1 + \frac{K}{2}\right)
\]
In other words, \(r_{ij}\) is marginally distributed according to the above Beta distribution scaled into \((-1, 1)\).
Value

- `dlkjcorr_marginal` gives the density
- `plkjcorr_marginal` gives the cumulative distribution function (CDF)
- `qlkjcorr_marginal` gives the quantile function (inverse CDF)
- `rlkjcorr_marginal` generates random draws.

The length of the result is determined by `n` for `rlkjcorr_marginal`, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than `n` are recycled to the length of the result. Only the first elements of the logical arguments are used.

References


See Also

`parse_dist()` and `marginalize_lkjcorr()` for parsing specs that use the LKJ correlation distribution and the `stat_slabinterval()` family of stats for visualizing them.

Examples

```r
library(dplyr)
library(ggplot2)
library(forcats)
theme_set(theme_ggdist())

expand.grid(
  eta = 1:6,
  K = 2:6
) %>%
ggplot(aes(y = fct_rev(ordered(eta)), dist = "lkjcorr_marginal", arg1 = K, arg2 = eta)) +
  stat_slab() +
  facet_grid(~ paste0(K, "x", K)) +
  labs(
    title = paste0(
      "Marginal correlation for LKJ(eta) prior on different matrix sizes:\n",
      "dlkjcorr_marginal(K, eta)"
    ),
    subtitle = "Correlation matrix size (KxK)",
    y = "eta",
    x = "Marginal correlation"
  ) +
  theme(axis.title = element_text(hjust = 0))
```
marginalize_lkjcorr

Turn spec for LKJ distribution into spec for marginal LKJ distribution

Description

Turns specs for an LKJ correlation matrix distribution as returned by `parse_dist()` into specs for the marginal distribution of a single cell in an LKJ-distributed correlation matrix (i.e., `lkjcorr_marginal()`). Useful for visualizing prior correlations from LKJ distributions.

Usage

```r
marginalize_lkjcorr(data, K, predicate = NULL, dist = ".dist", args = ".args")
```

Arguments

- `data`: A data frame containing a column with distribution names (".dist" by default) and a list column of distribution arguments (".args" by default), such as output by `parse_dist()`.
- `K`: Dimension of the correlation matrix. Must be greater than or equal to 2.
- `predicate`: A bare expression for selecting the rows of `data` to modify. This is useful if `data` contains more than one row with an LKJ prior in it and you only want to modify some of the distributions; if this is the case, give row a predicate expression (such as you might supply to `dplyr::filter()`) that evaluates to `TRUE` on the rows you want to modify. If `NULL` (the default), all `lkjcorr` distributions in `data` are modified.
- `dist`: The name of the column containing distribution names. See `parse_dist()`.
- `args`: The name of the column containing distribution arguments. See `parse_dist()`.

Details

The LKJ(\(\eta\)) prior on a correlation matrix induces a marginal prior on each correlation in the matrix that depends on both the value of \(\eta\) and \(K\), the dimension of the \(K \times K\) correlation matrix. Thus to visualize the marginal prior on the correlations, it is necessary to specify the value of \(K\), which depends on what your model specification looks like.

Given a data frame representing parsed distribution specifications (such as returned by `parse_dist()`), this function updates any rows with `.dist == "lkjcorr"` so that the first argument to the distribution is equal to the specified dimension of the correlation matrix \(K\) and changes the distribution name to "lkjcorr_marginal", allowing the distribution to be easily visualized using the `stat_slabinterval()` family of `ggplot2` stats.

Value

A data frame of the same size and column names as the input, with the `dist` and `args` columns modified on rows where `dist == "lkjcorr"` such that they represent a marginal LKJ correlation distribution with name `lkjcorr_marginal` and `args` having \(K\) equal to the input value of \(K\).
See Also

parse_dist(), lkjcorr_marginal()

Examples

library(dplyr)
library(ggplot2)

data.frame(prior = "lkjcorr(3)") %>%
  parse_dist(prior) %>%
  marginalize_lkjcorr(K = 2) %>%
  ggplot(aes(y = prior, dist = .dist, args = .args)) +
  stat_halfeye() +
  xlim(-1, 1) +
  xlab("Marginal correlation for LKJ(3) prior on 2x2 correlation matrix")

data.frame(coef = c("a", "b"), prior = "lkjcorr(3)") %>%
  parse_dist(prior) %>%
  marginalize_lkjcorr(K = 2, coef == "a") %>%
  marginalize_lkjcorr(K = 4, coef == "b")

parse_dist

Parse distribution specifications into columns of a data frame

Description

Parses simple string distribution specifications, like "normal(0, 1)", into two columns of a data frame, suitable for use with the dist and args aesthetics of stat_slabinterval() and its shortcut stats (like stat_halfeye()). This format is output by brms::get_prior, making it particularly useful for visualizing priors from brms models.

Usage

parse_dist(object, ..., dist = ".dist", args = ".args", to_r_names = TRUE)

## Default S3 method:
parse_dist(object, ...)

## S3 method for class 'data.frame'
parse_dist(
  object,
```r

dist_col,
..., 
dist = ".dist",
args = ".args",
to_r_names = TRUE
)

## S3 method for class 'character'
parse_dist(object, ..., dist = ".dist", args = ".args", to_r_names = TRUE)

## S3 method for class 'factor'
parse_dist(object, ..., dist = ".dist", args = ".args", to_r_names = TRUE)

## S3 method for class 'brmsprior'
parse_dist(
    object,
    dist_col = prior,
    ..., 
    dist = ".dist",
    args = ".args",
    to_r_names = TRUE
)

r_dist_name(dist_name)
```

**Arguments**

- `object` A character vector containing distribution specifications or a data frame with a column containing distribution specifications.
- `...` Arguments passed to other implementations of `parse_dist`.
- `dist` The name of the output column to contain the distribution name
- `args` The name of the output column to contain the arguments to the distribution
- `to_r_names` If TRUE (the default), certain common aliases for distribution names are automatically translated into names that R can recognize (i.e., names which have functions starting with `r`, `p`, `q`, and `d` representing random number generators, distribution functions, etc. for that distribution), using the `r_dist_name` function. For example, "normal" is translated into "norm" and "lognormal" is translated into "lnorm".
- `dist_col` A bare (unquoted) column or column expression that resolves to a character vector of distribution specifications.
- `dist_name` For `r_dist_name`, a character vector of distribution names to be translated into distribution names R recognizes. Unrecognized names are left as-is.

**Details**

`parse_dist()` can be applied to character vectors or to a data frame + bare column name of the column to parse, and returns a data frame with ".dist" and ".args" columns added. `parse_dist()` uses `r_dist_name()` to translate distribution names into names recognized by R.
**r_dist_name()** takes a character vector of names and translates common names into R distribution names. Names are first made into valid R names using `make.names()`, then translated (ignoring character case, ",", and "."). Thus, "lognormal", "LogNormal", "log_normal", "log-Normal", and any number of other variants all get translated into "lnorm".

**Value**

- `parse_dist` returns a data frame containing at least two columns named after the `dist` and `args` parameters. If the input is a data frame, the output is a data frame of the same length with those two columns added. If the input is a character vector or factor, the output is a two-column data frame with the same number of rows as the length of the input.
- `r_dist_name` returns a character vector the same length as the input containing translations of the input names into distribution names R can recognize.

**See Also**

See `stat_slabinterval()` and its shortcut stats, which can easily make use of the output of this function using the `dist` and `args` aesthetics.

**Examples**

```r
library(dplyr)

# parse dist can operate on strings directly...
parse_dist(c("normal(0,1)", "student_t(3,0,1)"))

# ... or on columns of a data frame, where it adds the
# parsed specs back on as columns
data.frame(prior = c("normal(0,1)", "student_t(3,0,1)")) %>%
  parse_dist(prior)

# parse_dist is particularly useful with the output of brms::prior(),
# which follow the same format as above
```
point_interval

Usage

point_interval(
  .data,
  ..., 
  .width = 0.95,
  .point = median,
  .interval = qi,
  .simple_names = TRUE,
  na.rm = FALSE,
  .exclude = c(".chain", ".iteration", ".draw", ".row"),
  .prob
)

## Default S3 method:
point_interval(
  .data,
  ..., 
  .width = 0.95,
  .point = median,
  .interval = qi,
  .simple_names = TRUE,
  na.rm = FALSE,
  .exclude = c(".chain", ".iteration", ".draw", ".row"),
  .prob
)

## S3 method for class 'numeric'
point_interval(
  .data,
  ..., 
  .width = 0.95,
  .point = median,
  .interval = qi,
  .simple_names = FALSE,
  na.rm = FALSE,
  .exclude = c(".chain", ".iteration", ".draw", ".row"),
  .prob
)

## S3 method for class 'rvar'
point_interval(
  .data,
  ..., 
  .width = 0.95,
  .point = median,
  .interval = qi,
  .simple_names = TRUE,
  na.rm = FALSE
)
point_interval

## S3 method for class 'distribution'
point_interval(
  .data,
  ..., 
  .width = 0.95, 
  .point = median, 
  .interval = qi, 
  .simple_names = TRUE, 
  na.rm = FALSE 
)

qi(x, .width = 0.95, .prob, na.rm = FALSE)

ll(x, .width = 0.95, na.rm = FALSE)

ul(x, .width = 0.95, na.rm = FALSE)

hdi(x, .width = 0.95, .prob, na.rm = FALSE, ...)

Mode(x, na.rm = FALSE)

## Default S3 method:
Mode(x, na.rm = FALSE)

## S3 method for class 'rvar'
Mode(x, na.rm = FALSE)

## S3 method for class 'distribution'
Mode(x, na.rm = FALSE)

hdci(x, .width = 0.95, na.rm = FALSE)

mean_qi(.data, ..., .width = 0.95)

median_qi(.data, ..., .width = 0.95)

mode_qi(.data, ..., .width = 0.95)

mean_ll(.data, ..., .width = 0.95)

median_ll(.data, ..., .width = 0.95)

mode_ll(.data, ..., .width = 0.95)

mean_ul(.data, ..., .width = 0.95)
point_interval

median_ul(.data, ..., .width = 0.95)
mode_ul(.data, ..., .width = 0.95)
mean_hdi(.data, ..., .width = 0.95)
median_hdi(.data, ..., .width = 0.95)
mode_hdi(.data, ..., .width = 0.95)
mean_hdci(.data, ..., .width = 0.95)
median_hdci(.data, ..., .width = 0.95)
mode_hdci(.data, ..., .width = 0.95)

Arguments

.data Data frame (or grouped data frame as returned by group_by()) that contains draws to summarize.

... Bare column names or expressions that, when evaluated in the context of .data, represent draws to summarize. If this is empty, then by default all columns that are not group columns and which are not in .exclude (by default ".chain", ".iteration", ".draw", and ".row") will be summarized. These columns can be numeric, distributional objects, posterior::rvars, or list columns of numeric values to summarise.

.width vector of probabilities to use that determine the widths of the resulting intervals. If multiple probabilities are provided, multiple rows per group are generated, each with a different probability interval (and value of the corresponding .width column).

.point Point summary function, which takes a vector and returns a single value, e.g. mean(), median(), or Mode().

.interval Interval function, which takes a vector and a probability (.width) and returns a two-element vector representing the lower and upper bound of an interval; e.g. qi(), hdi()}

.simple_names When TRUE and only a single column / vector is to be summarized, use the name .lower for the lower end of the interval and .upper for the upper end. If .data is a vector and this is TRUE, this will also set the column name of the point summary to .value. When FALSE and .data is a data frame, names the lower and upper intervals for each column x.lower and x.upper. When FALSE and .data is a vector, uses the naming scheme y, ymin and ymax (for use with ggplot).

.na.rm logical value indicating whether NA values should be stripped before the computation proceeds. If FALSE (the default), any vectors to be summarized that contain NA will result in point and interval summaries equal to NA.

.exclude A character vector of names of columns to be excluded from summarization if no column names are specified to be summarized. Default ignores several
meta-data column names used in tidybayes.

- `.prob` Deprecated. Use `.width` instead.
- `x` vector to summarize (for interval functions: `qi` and `hdi`)

**Details**

If `.data` is a data frame, then ... is a list of bare names of columns (or expressions derived from columns) of `.data`, on which the point and interval summaries are derived. Column expressions are processed using the tidy evaluation framework (see `rlang::eval_tidy()`).

For a column named `x`, the resulting data frame will have a column named `x` containing its point summary. If there is a single column to be summarized and `.simple_names` is `TRUE`, the output will also contain columns `.lower` (the lower end of the interval), `.upper` (the upper end of the interval). Otherwise, for every summarized column `x`, the output will contain `x`.lower (the lower end of the interval) and `x`.upper (the upper end of the interval). Finally, the output will have a `.width` column containing the probability for the interval on each output row.

If `.data` includes groups (see e.g. `dplyr::group_by()`), the points and intervals are calculated within the groups.

If `.data` is a vector, ... is ignored and the result is a data frame with one row per value of `.width` and three columns: `y` (the point summary), `ymin` (the lower end of the interval), `ymax` (the upper end of the interval), and `.width`, the probability corresponding to the interval. This behavior allows `point_interval` and its derived functions (like `median_qi`, `mean_qi`, `mode_hdi`, etc) to be easily used to plot intervals in ggplot stats using methods like `stat_eye()`, `stat_halfeye()`, or `stat_summary()`.

`median_qi`, `mode_hdi`, etc are short forms for `point_interval(..., .point = median, .interval = qi)`, etc.

`qi` yields the quantile interval (also known as the percentile interval or equi-tailed interval) as a 1x2 matrix.

`hdi` yields the highest-density interval(s) (also known as the highest posterior density interval). **Note:** If the distribution is multimodal, `hdi` may return multiple intervals for each probability level (these will be spread over rows). You may wish to use `hdci` (below) instead if you want a single highest-density interval, with the caveat that when the distribution is multimodal `hdci` is not a highest-density interval. Internally `hdi` uses `HDIInterval::hdi()` with `allowSplit = TRUE` (when multimodal) and with `allowSplit = FALSE` (when not multimodal).

`hdci` yields the highest-density continuous interval. **Note:** If the distribution is multimodal, this may not actually be the highest-density interval (there may be a higher-density discontinuous interval). Internally `hdci` uses `HDIInterval::hdi()` with `allowSplit = FALSE`; see that function for more information on multimodality and continuous versus discontinuous intervals.

`ll` and `ul` yield lower limits and upper limits, respectively (where the opposite limit is set to either `Inf` or `-Inf`).

**Value**

A data frame containing point summaries and intervals, with at least one column corresponding to the point summary, one to the lower end of the interval, one to the upper end of the interval, the width of the interval `.width`, the type of point summary `.point`, and the type of interval `.interval`. 
position_dodgejust

Author(s)

Matthew Kay

Examples

library(dplyr)
library(ggplot2)

set.seed(123)
rnorm(1000) %>%
  median_qi()

data.frame(x = rnorm(1000)) %>%
  median_qi(x, .width = c(.50, .80, .95))

data.frame(
  x = rnorm(1000),
  y = rnorm(1000, mean = 2, sd = 2)
) %>%
  median_qi(x, y)

data.frame(
  x = rnorm(1000),
  group = "a"
) %>%
  rbind(data.frame(
    x = rnorm(1000, mean = 2, sd = 2),
    group = "b"
  )) %>%
  group_by(group) %>%
  median_qi(.width = c(.50, .80, .95))

multimodal_draws = data.frame(
  x = c(rnorm(5000, 0, 1), rnorm(2500, 4, 1))
)

multimodal_draws %>%
  mode_hdi(.width = c(.66, .95))

multimodal_draws %>%
  ggplot(aes(x = x, y = 0)) +
  stat_halfeye(point_interval = mode_hdi, .width = c(.66, .95))

---

position_dodgejust  Dodge overlapping objects side-to-side, preserving justification
Description

A justification-preserving variant of `ggplot2::position_dodge()` which preserves the vertical position of a geom while adjusting the horizontal position (or vice versa when in a horizontal orientation). Unlike `ggplot2::position_dodge()`, `position_dodgejust()` attempts to preserve the "justification" of x positions relative to the bounds containing them (xmin/xmax) (or y positions relative to ymin/ymax when in a horizontal orientation). This makes it useful for dodging annotations to geoms and stats from the `geom_slabinterval()` family, which also preserve the justification of their intervals relative to their slabs when dodging.

Usage

```r
position_dodgejust(
  width = NULL,
  preserve = c("total", "single"),
  justification = NULL
)
```

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?
- **justification**: Justification of the point position (x/y) relative to its bounds (xmin/xmax or ymin/ymax), where 0 indicates bottom/left justification and 1 indicates top/right justification (depending on orientation). This is only used if xmin/xmax/ymin/ymax are not supplied; in that case, justification will be used along with width to determine the bounds of the object prior to dodging.

Examples

```r
library(dplyr)
library(ggplot2)
library(distributional)

dist_df = tribble(~group, ~subgroup, ~mean, ~sd,
  1, "h", 5, 1,
  2, "h", 7, 1.5,
  3, "h", 8, 1,
  3, "i", 9, 1,
  3, "j", 7, 1
)

# An example with normal "dodge" positioning
# Notice how dodge points are placed in the center of their bounding boxes,
# which can cause slabs to be positioned outside their bounds.
```
position_dodgejust

```r
dist_df %>%
  ggplot(aes(
    x = factor(group), ydist = dist_normal(mean, sd),
    fill = subgroup
  )) +
  stat_halfeye(
    position = "dodge"
  ) +
  geom_rect(
    aes(xmin = group, xmax = group + 1, ymin = 2, ymax = 13, color = subgroup),
    position = "dodge",
    data = . %>% filter(group == 3),
    alpha = 0.1
  ) +
  geom_point(
    aes(x = group, y = 7.5, color = subgroup),
    position = position_dodgejust(width = 1),
    data = . %>% filter(group == 3),
    shape = 1,
    size = 4,
    stroke = 1.5
  ) +
  scale_fill_brewer(palette = "Set2") +
  scale_color_brewer(palette = "Dark2")

# This same example with "dodgejust" positioning. For the points we
# supply a justification parameter to position_dodgejust which mimics the
# justification parameter of stat_halfeye, ensuring that they are
# placed appropriately. On slabinterval family geoms, position_dodgejust()
# will automatically detect the appropriate justification.

```
```
scale_color_brewer(palette = "Dark2")

---

**scales**

Custom ggplot scales for geom_slabinterval (and derivatives)

**Description**

These scales allow more specific aesthetic mappings to be made when using `geom_slabinterval()` and stats/geoms based on it (like eye plots).

**Usage**

```r
scale_point_colour_discrete(..., aesthetics = "point_colour")

scale_point_color_discrete(..., aesthetics = "point_colour")

scale_point_colour_continuous(
  ..., 
  aesthetics = "point_colour", 
  guide = guide_colourbar2()
)

scale_point_color_continuous(
  ..., 
  aesthetics = "point_colour", 
  guide = guide_colourbar2()
)

scale_point_fill_discrete(..., aesthetics = "point_fill")

scale_point_fill_continuous(
  ..., 
  aesthetics = "point_fill", 
  guide = guide_colourbar2()
)

scale_point_alpha_continuous(..., range = c(0.1, 1))

scale_point_alpha_discrete(..., range = c(0.1, 1))

scale_point_size_continuous(..., range = c(1, 6))

scale_point_size_discrete(..., range = c(1, 6), na.translate = FALSE)
```
scale_interval_colour_discrete(..., aesthetics = "interval_colour")
scale_interval_color_discrete(..., aesthetics = "interval_colour")

scale_interval_colour_continuous(
    ..., 
    aesthetics = "interval_colour", 
    guide = guide_colourbar2()
)
scale_interval_color_continuous(
    ..., 
    aesthetics = "interval_colour", 
    guide = guide_colourbar2()
)
scale_interval_alpha_continuous(..., range = c(0.1, 1))
scale_interval_alpha_discrete(..., range = c(0.1, 1))
scale_interval_size_continuous(..., range = c(1, 6))
scale_interval_size_discrete(..., range = c(1, 6), na.translate = FALSE)
scale_interval_linetype_discrete(..., na.value = "blank")
scale_interval_linetype_continuous(...)
scale_slab_colour_discrete(..., aesthetics = "slab_colour")
scale_slab_color_discrete(..., aesthetics = "slab_colour")

scale_slab_colour_continuous(
    ..., 
    aesthetics = "slab_colour", 
    guide = guide_colourbar2()
)
scale_slab_color_continuous(
    ..., 
    aesthetics = "slab_colour", 
    guide = guide_colourbar2()
)
scale_slab_fill_discrete(..., aesthetics = "slab_fill")
scale_slab_fill_continuous(
    ..., 
    ...,
aesthetics = "slab_fill",
guide = guide_colourbar2()
)

scale_slab_alpha_continuous(
  ...,
  limits = function(l) c(min(0, l[[1]]), l[[2]]),
  range = c(0, 1)
)

scale_slab_alpha_discrete(..., range = c(0.1, 1))

scale_slab_size_continuous(..., range = c(1, 6))

scale_slab_size_discrete(..., range = c(1, 6), na.translate = FALSE)

scale_slab_linetype_discrete(..., na.value = "blank")

scale_slab_linetype_continuous(...)

scale_slab_shape_discrete(..., solid = TRUE)

scale_slab_shape_continuous(...)

guide_colourbar2(...)

guide_colorbar2(...)

Arguments

... Arguments passed to underlying scale or guide functions. E.g. scale_point_color_discrete passes arguments to scale_color_discrete(). See those functions for more details.

aesthetics Names of aesthetics to set scales for.

guide Guide to use for legends for an aesthetic.

range a numeric vector of length 2 that specifies the minimum and maximum size of the plotting symbol after transformation.

na.translate In discrete scales, should we show missing values?

na.value When na.translate is true, what value should be shown?

limits One of:

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

**solid**  Should the shapes be solid, TRUE, or hollow, FALSE?

**Details**

The following additional scales / aesthetics are defined for use with `geom_slabinterval()` and related geoms:

1. `scale_point_color_*`  Point color
2. `scale_point_fill_*`  Point fill color
3. `scale_point_alpha_*`  Point alpha level / opacity
4. `scale_point_size_*`  Point size
5. `scale_interval_color_*`  Interval line color
6. `scale_interval_alpha_*`  Interval alpha level / opacity
7. `scale_interval_size_*`  Interval line width
8. `scale_interval_linetype_*`  Interval line type
9. `scale_slab_color_*`  Slab outline color
10. `scale_slab_fill_*`  Slab fill color
11. `scale_slab_alpha_*`  Slab alpha level / opacity. The default settings of `scale_slab_alpha_continuous` differ from `scale_alpha_continuous()` and are designed for gradient plots (e.g. `stat_gradientinterval()`) by ensuring that densities of 0 get mapped to 0 in the output.
12. `scale_slab_size_*`  Slab outline line width
13. `scale_slab_linetype_*`  Slab outline line type
14. `scale_slab_shape_*`  Slab dot shape (for `geom_dotsinterval()`)

See the corresponding scale documentation in ggplot for more information; e.g. `scale_color_discrete()`, `scale_color_continuous()`, etc.

Other scale functions can be used with the aesthetics/scales defined here by using the aesthetics argument to that scale function. For example, to use color brewer scales with the point_color aesthetic:

```
scale_color_brewer(..., aesthetics = "point_color")
```

With continuous color scales, you may also need to provide a guide as the default guide does not work properly; this is what `guide_colorbar2` is for:

```
scale_color_distiller(..., guide = "colorbar2", aesthetics = "point_color")
```

**Value**

A `ggplot2::Scale` representing one of the aesthetics used to target the appearance of specific parts of composite `ggdist` geoms. Can be added to a `ggplot()` object.

**Author(s)**

Matthew Kay
scale_colour_ramp

See Also

Other ggplot2 scales: `scale_color_discrete()`, `scale_color_continuous()`, etc.

Other ggdist scales: `scale_colour_ramp`, `scale_thickness`

Examples

```r
library(dplyr)
library(ggplot2)

# This plot shows how to set multiple specific aesthetics
# NB it is very ugly and is only for demo purposes.
data.frame(distribution = "Normal(1,2)" ) %>%
  parse_dist(distribution) %>%
ggplot(aes(y = distribution, xdist = .dist, args = .args)) +
  stat_halfeye(
    shape = 21, # this point shape has a fill and outline
    point_color = "red",
    point_fill = "black",
    point_alpha = .1,
    point_size = 6,
    stroke = 2,
    interval_color = "blue",
    # interval sizes are scaled from [1, 6] onto [0.6, 1.4] by default
    # see the interval_size_range parameter in help("geom_slabinterval")
    interval_size = 8,
    interval_linetype = "dashed",
    interval_alpha = .25,
    # fill sets the fill color of the slab (here the density)
    slab_color = "green",
    slab_fill = "purple",
    slab_size = 3,
    slab_linetype = "dotted",
    slab_alpha = .5
  )
```

Description

This scale creates a secondary scale that modifies the fill or color scale of geoms that support it (`geom_lineribbon()` and `geom_slabinterval()`) to "ramp" from a secondary color (by default white) to the primary fill color (determined by the standard color or fill aesthetics).
scale_colour_ramp

Usage

```r
scale_colour_ramp_continuous(
  from = "white",
  ...
)

scale_color_ramp_continuous(
  from = "white",
  ...
)

scale_colour_ramp_discrete(
  from = "white",
  ...
)

scale_color_ramp_discrete(
  from = "white",
  ...
)

scale_fill_ramp_continuous(..., aesthetics = "fill_ramp")

scale_fill_ramp_discrete(..., aesthetics = "fill_ramp")
```

Arguments

- **from**: The color to ramp from. Corresponds to 0 on the scale.
- **...**: Arguments passed to underlying scale or guide functions. E.g. `scale_colour_ramp_discrete()`, passes arguments to `discrete_scale()`, `scale_colour_ramp_continuous()` passes arguments to `continuous_scale()`. See those functions for more details.
- **limits**: One of:
  - NULL to use the default scale range
• A numeric vector of length two providing limits of the scale. Use NA to refer to the existing minimum or maximum
• A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang `lambda` function notation. Note that setting limits on positional scales will remove data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see `coord_cartesian()`).

range
a numeric vector of length 2 that specifies the minimum and maximum values after the scale transformation. These values should be between 0 (the from color) and 1 (the color determined by the fill aesthetic).

guide
A function used to create a guide or its name. For `scale_colour_ramp_continuous()` and `scale_fill_ramp_continuous()`, `guide_rampbar()` can be used to create gradient color bars. See `guides()` for information on other guides.

aesthetics
Names of aesthetics to set scales for.

Value
A `ggplot2::Scale` representing a scale for the `colour_ramp` and/or `fill_ramp` aesthetics for `ggdist` geoms. Can be added to a `ggplot()` object.

Author(s)
Matthew Kay

See Also
`guide_rampbar()`

Other `ggdist` scales: `scale_thickness, scales`

Examples

```r
library(dplyr)
library(ggplot2)
library(distributional)

tibble(d = dist_uniform(0, 1)) %>%
  ggplot(aes(y = 0, xdist = d)) +
  stat_slab(aes(fill_ramp = stat(x)))

tibble(d = dist_uniform(0, 1)) %>%
  ggplot(aes(y = 0, xdist = d)) +
  stat_slab(aes(fill_ramp = stat(x)), fill = "blue") +
  scale_fill_ramp_continuous(from = "red")

# you can invert the order of `range` to change the order of the blend

tibble(d = dist_normal(0, 1)) %>%
  ggplot(aes(y = 0, xdist = d)) +
  stat_slab(aes(fill_ramp = stat(cut_cdf_qi(cdf))), fill = "blue") +
  scale_fill_ramp_discrete(from = "red", range = c(1, 0))
```
This \texttt{ggplot2} scale linearly scales all thickness values of geoms that support the \texttt{thickness} aesthetic (such as \texttt{geom_slabinterval()}). It can be used to align the \texttt{thickness} scales across multiple geoms (by default, \texttt{thickness} is normalized on a per-geom level instead of as a global scale).

\textbf{Usage}

\begin{verbatim}
scale_thickness_shared(
    name = waiver(),
    breaks = waiver(),
    labels = waiver(),
    limits = function(l) c(min(0, l[[1]]), l[[2]]),
    renormalize = FALSE,
    guide = "none"
)

scale_thickness_identity(..., guide = "none")

thickness(x = double())
\end{verbatim}

\textbf{Arguments}

- **name**: The name of the scale. Used as the axis or legend title. If \texttt{waiver()}, the default, the name of the scale is taken from the first mapping used for that aesthetic. If \texttt{NULL}, the legend title will be omitted.

- **breaks**: One of:
  - \texttt{NULL} for no breaks
  - \texttt{waiver()} for the default breaks computed by the \texttt{transformation object}
  - A numeric vector of positions
  - A function that takes the limits as input and returns breaks as output (e.g., a function returned by \texttt{scales::extended_breaks()}). Also accepts rlang \texttt{lambda} function notation.

- **labels**: One of:
  - \texttt{NULL} for no labels
  - \texttt{waiver()} for the default labels computed by the \texttt{transformation object}
  - A character vector giving labels (must be same length as \texttt{breaks})
  - A function that takes the breaks as input and returns labels as output. Also accepts rlang \texttt{lambda} function notation.

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

`renormalize`  When mapping values to the thickness scale, should those values be allowed to be renormalized by geoms (e.g. via the normalize parameter to `geom_slabinterval()`)? The default is FALSE: if `scale_thickness_shared()` is in use, the geom-specific normalize parameter is ignored (this is achieved by flagging values as already normalized by wrapping them in `thickness()`). Set this to TRUE to allow geoms to also apply their own normalization.

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

`...`  Arguments passed to the underlying scale or guide functions. E.g. `scale_thickness_identity()` passes arguments to `continuous_scale()`. See that function for more details.

`x`  An object (typically a numeric()) to be converted to a thickness() object.

**Details**

By default, normalization/scaling of slab thicknesses is controlled by geometries, not by a `ggplot2` scale function. This allows various functionality not otherwise possible, such as (1) allowing different geometries to have different thickness scales and (2) allowing the user to control at what level of aggregation (panels, groups, the entire plot, etc) thickness scaling is done via the normalize parameter to `geom_slabinterval()`.

However, this default approach has one drawback: two different geoms will always have their own scaling of thickness. `scale_thickness_shared()` offers an alternative approach: when added to a chart, all geoms will use the same thickness scale, and geom-level normalization (via their normalize parameters) is ignored. This is achieved by "marking" thickness values as already normalized by wrapping them in the thickness() data type (this can be disabled by setting renormalize = TRUE).

`thickness()` is used by `scale_thickness_shared()` to create numeric()-like objects marked as being in units of slab "thickness". Unlike regular numeric(), thickness() values mapped onto the thickness aesthetic are not rescaled by `scale_thickness_shared()` or `geom_slabinterval()`. In most cases thickness() is not useful directly; though it can be used to mark values that should not be rescaled—see the definitions of `stat_ccdfinterval()` and `stat_gradientinterval()` for some usages.

Note: while a slightly more typical name for `scale_thickness_shared()` might be `scale_thickness_continuous()`, the latter name would cause this scale to be applied to all thickness aesthetics by default according to the rules `ggplot2` uses to find default scales. Thus, to retain the usual behavior of `stat_slabinterval()` (per-geom normalization of thickness), this scale is called `scale_thickness_shared()`.
Value

A `ggplot2::Scale` representing a scale for the thickness aesthetic for `ggdist` geoms. Can be added to a `ggplot()` object.

Author(s)

Matthew Kay

See Also

The thickness aesthetic of `geom_slabinterval()`.

Other `ggdist` scales: `scale_colour_ramp`, `scales`

Examples

```r
library(distributional)
library(ggplot2)
library(dplyr)

prior_post = data.frame(
  prior = dist_normal(0, 1),
  posterior = dist_normal(0.1, 0.5)
)

# By default, separate geoms have their own thickness scales, which means
# distributions plotted using two separate geoms will not have their slab
# functions drawn on the same scale (thus here, the two distributions have
# different areas under their density curves):
prior_post %>%
  ggplot() +
  stat_halfeye(aes(xdist = posterior)) +
  stat_slab(aes(xdist = prior), fill = NA, color = "red")

# For this kind of prior/posterior chart, it makes more sense to have the
# densities on the same scale; thus, the areas under both would be the same.
# We can do that using `scale_thickness_shared()`:
prior_post %>%
  ggplot() +
  stat_halfeye(aes(xdist = posterior)) +
  stat_slab(aes(xdist = prior), fill = NA, color = "#e41a1c") +
  scale_thickness_shared()
```
**Description**

Shortcut version of `stat_slabinterval()` with `geom_slabinterval()` for creating CCDF bar plots.

Roughly equivalent to:

```r
stat_slabinterval(
  aes(thickness = stat(thickness(1 - cdf)), justification = stat(0.5), side = stat("topleft")),
  slab_type = "ccdf", normalize = "none", expand = TRUE
)
```

**Usage**

```r
stat_ccdfinterval(
  mapping = NULL,
  data = NULL,
  geom = "slabinterval",
  position = "identity",
  ...
  slab_type = "ccdf",
  normalize = "none",
  expand = TRUE,
  p_limits = c(NA, NA),
  adjust = 1,
  trim = TRUE,
  breaks = "Sturges",
  outline_bars = FALSE,
  point_interval = "median_qi",
  limits = NULL,
  n = 501,
  .width = c(0.66, 0.95),
  orientation = NA,
  na.rm = FALSE,
  show.legend = c(size = FALSE),
  inherit.aes = TRUE
)
```

**Arguments**

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

- **data**
  The data to be displayed in this layer. There are three options:
  - If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
  - A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. \( \sim \) head(.x, 10)).

**geom**

Use to override the default connection between `stat_ccdfinterval()` and `geom_slabinterval()`.

**position**

Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" (`position_dodge()`) or "dodgejust" (`position_dodgejust()`) can be useful if you have overlapping geometries.

... Other arguments passed to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3` (see Aesthetics, below). They may also be parameters to the paired geom/stat. When paired with the default geom, `geom_slabinterval()`, these include:

**fill_type** What type of fill to use when the fill color or alpha varies within a slab. One of:

- "segments": breaks up the slab geometry into segments for each unique combination of fill color and alpha value. This approach is supported by all graphics devices and works well for sharp cutoff values, but can give ugly results if a large number of unique fill colors are being used (as in gradients, like in `stat_gradientinterval()`).
- "gradient": a `grid::linearGradient()` is used to create a smooth gradient fill. This works well for large numbers of unique fill colors, but requires R >= 4.1 and is not yet supported on all graphics devices. As of this writing, the `png()` graphics device with type = "cairo", the `svg()` device, the `pdf()` device, and the `ragg::agg_png()` devices are known to support this option. On R < 4.1, this option will fall back to `fill_type = "segments"` with a message.
- "auto": attempts to use `fill_type = "gradient"` if support for it can be auto-detected. On R >= 4.2, support for gradients can be auto-detected on some graphics devices; if support is not detected, this option will fall back to `fill_type = "segments"` (in case of a false negative, `fill_type = "gradient"` can be set explicitly). On R < 4.2, support for gradients cannot be auto-detected, so this will always fall back to `fill_type = "segments"`, in which case you can set `fill_type = "gradient"` explicitly if you are using a graphics device that support gradients.

**interval_size_domain** A length-2 numeric vector giving the minimum and maximum of the values of the size aesthetic that will be translated into actual sizes for intervals drawn according to `interval_size_range` (see the documentation for that argument.)

**interval_size_range** A length-2 numeric vector. This geom scales the raw size aesthetic values when drawing interval and point sizes, as they tend to be too thick when using the default settings of `scale_size_continuous()`, which give sizes with a range of `c(1, 6)`. The `interval_size_domain` value indicates the input domain of raw size values (typically this should be equal to the value of the range argument of the `scale_size_continuous()` function), and `interval_size_range` indicates the desired output range of
the size values (the min and max of the actual sizes used to draw intervals). Most of the time it is not recommended to change the value of this argument, as it may result in strange scaling of legends; this argument is a holdover from earlier versions that did not have size aesthetics targeting the point and interval separately. If you want to adjust the size of the interval or points separately, you can instead use the interval_size or point_size aesthetics; see scales.

fatten_point  A multiplicative factor used to adjust the size of the point relative to the size of the thickest interval line. If you wish to specify point sizes directly, you can also use the point_size aesthetic and scale_point_size_continuous() or scale_point_size_discrete(); sizes specified with that aesthetic will not be adjusted using fatten_point.

slab_type  The type of slab function to calculate: probability density (or mass) function ("pdf"), cumulative distribution function ("cdf"), or complementary CDF ("ccdf").

normalize  How to normalize heights of functions input to the thickness aesthetic. One of:

- "all": normalize so that the maximum height across all data is 1.
- "panels": normalize within panels so that the maximum height in each panel is 1.
- "xy": normalize within the x/y axis opposite the orientation of this geom so that the maximum height at each value of the opposite axis is 1.
- "groups": normalize within values of the opposite axis and within each group so that the maximum height in each group is 1.
- "none": values are taken as is with no normalization (this should probably only be used with functions whose values are in [0,1], such as CDFs).

expand  For sample data, should the slab be expanded to the limits of the scale? Default FALSE. Can be length two to control expansion to the lower and upper limit respectively.

p_limits  Probability limits (as a vector of size 2) used to determine the lower and upper limits of the slab. E.g., if this is c(.001, .999), then a slab is drawn for the distribution from the quantile at p = .001 to the quantile at p = .999. If the lower (respectively upper) limit is NA, then the lower (upper) limit will be the minimum (maximum) of the distribution’s support if it is finite, and 0 .001 (0.999) if it is not finite. E.g., if p_limits is c(NA, NA) on a gamma distribution the effective value of p_limits would be c(0, .999) since the gamma distribution is defined on (0, Inf); whereas on a normal distribution it would be equivalent to c(.001, .999) since the normal distribution is defined on (-Inf, Inf).

adjust  If slab_type is "pdf", bandwidth for the density estimator for sample data is adjusted by multiplying it by this value. See density() for more information.

trim  For sample data, should the density estimate be trimmed to the range of the input data? Default TRUE.

breaks  If slab_type is "histogram", the breaks parameter that is passed to hist() to determine where to put breaks in the histogram (for sample data).

outline_bars  For sample data (if slab_type is "histogram") and for discrete analytical distributions (whose slabs are drawn as histograms), determines if outlines in between the bars are drawn when the slab_color aesthetic is used. If FALSE (the
default), the outline is drawn only along the tops of the bars; if TRUE, outlines in between bars are also drawn.

**point_interval**  A function from the `point_interval()` family (e.g., `median_qi`, `mean_qi`, `mode_hdi`, etc), or a string giving the name of a function from that family (e.g., "median_qi", "mean_qi", "mode_hdi", etc; if a string, the caller’s environment is searched for the function, followed by the `ggdist` environment). This function determines the point summary (typically mean, median, or mode) and interval type (quantile interval, qi; highest-density interval, hdi; or highest-density continuous interval, hdci). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of `orientation`. See the `point_interval()` family of functions for more information.

**limits**  Manually-specified limits for the slab, as a vector of length two. These limits are combined with those computed based on `p_limits` as well as the limits defined by the scales of the plot to determine the limits used to draw the slab functions: these limits specify the maximal limits; i.e., if specified, the limits will not be wider than these (but may be narrower). Use NA to leave a limit alone; e.g. `limits = c(0, NA)` will ensure that the lower limit does not go below 0, but let the upper limit be determined by either `p_limits` or the scale settings.

**n**  Number of points at which to evaluate the function that defines the slab.

**.width**  The `.width` argument passed to `point_interval`: a vector of probabilities to use that determine the widths of the resulting intervals. If multiple probabilities are provided, multiple intervals per group are generated, each with a different probability interval (and value of the corresponding `.width` and `level` generated variables).

**orientation**  Whether this geom is drawn horizontally or vertically. One of:

- NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
- "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify different groups. For each group, uses the x, xmin, xmax, and thickness aesthetics to draw points, intervals, and slabs.
- "vertical" (or "x"): draw vertically, using the x aesthetic to identify different groups. For each group, uses the y, ymin, ymax, and thickness aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for `orientation`, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal" (tidybayes had an orientation parameter before base ggplot did, hence the discrepancy).

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

**show.legend**  Should this layer be included in the legends? Default is `c(size = FALSE)`, unlike most geoms, to match its common use cases. FALSE hides all legends, TRUE shows all legends, and NA shows only those that are mapped (the default for most geoms).

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

To visualize sample data, such as a data distribution, samples from a bootstrap distribution, or a Bayesian posterior, you can supply samples to the x or y aesthetic.

To visualize analytical distributions, you can use the xdist or ydist aesthetic. For historical reasons, you can also use dist to specify the distribution, though this is not recommended as it does not work as well with orientation detection. These aesthetics can be used as follows:

- xdist, ydist, and dist can be any distribution object from the distributional package (dist_normal(), dist_beta(), etc) or can be a posterior::rvar() object. Since these functions are vectorized, other columns can be passed directly to them in an aes() specification; e.g. aes(dist = dist_normal(mu, sigma)) will work if mu and sigma are columns in the input data frame.
- dist can be a character vector giving the distribution name. Then the arg1, ... arg9 aesthetics (or args as a list column) specify distribution arguments. Distribution names should correspond to R functions that have "p", "q", and "d" functions; e.g. "norm" is a valid distribution name because R defines the pnorm(), qnorm(), and dnorm() functions for Normal distributions.

See the parse_dist() function for a useful way to generate dist and args values from human-readable distribution specs (like "normal(0,1)"). Such specs are also produced by other packages (like the brms::get_prior function in brms); thus, parse_dist() combined with the stats described here can help you visualize the output of those functions.

Value

A ggplot2::Stat representing a CCDF bar geometry which can be added to a ggplot() object.

Computed Variables

The following variables are computed by this stat and made available for use in aesthetic specifications (aes()) using the stat() or after_stat() functions:

- x or y: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is x or y depends on orientation
- xmin or ymin: For intervals, the lower end of the interval from the interval function.
- xmax or ymax: For intervals, the upper end of the interval from the interval function.
- .width: For intervals, the interval width as a numeric value in [0, 1]. For slabs, the width of the smallest interval containing that value of the slab.
- .level: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.
- pdf: For slabs, the probability density function (PDF). If options("ggdist.experimental.slab_data_in_intervals") is TRUE: For intervals, the PDF at the point summary; intervals also have pdf_min and pdf_max for the PDF at the lower and upper ends of the interval.
- cdf: For slabs, the cumulative distribution function. If options("ggdist.experimental.slab_data_in_intervals") is TRUE: For intervals, the CDF at the point summary; intervals also have cdf_min and cdf_max for the CDF at the lower and upper ends of the interval.
- f: For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by slab_type.
- n: For slabs, the number of data points summarized into that slab. If the slab was created from an analytical distribution via the xdist, ydist, or dist aesthetic, n will be Inf.
The slab+interval stats and geoms have a wide variety of aesthetics that control the appearance of their three sub-geometries: the slab, the point, and the interval.

These stats support the following aesthetics:

- **x**: x position of the geometry (when orientation = "vertical"); or sample data to be summarized (when orientation = "horizontal" with sample data).
- **y**: y position of the geometry (when orientation = "horizontal"); or sample data to be summarized (when orientation = "vertical" with sample data).
- **xdist**: When using analytical distributions, distribution to map on the x axis: a distributional object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- **ydist**: When using analytical distributions, distribution to map on the y axis: a distributional object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- **dist**: When using analytical distributions, a name of a distribution (e.g. "norm"), a distributional object (e.g. `dist_normal()`), or a `posterior::rvar()` object. See Details.
- **args**: Distribution arguments (args or arg1, ... arg9). See Details.

In addition, in their default configuration (paired with `geom_slabinterval()`) the following aesthetics are supported by the underlying geom:

### Slab-specific aesthetics

- **thickness**: The thickness of the slab at each x value (if orientation = "horizontal") or y value (if orientation = "vertical") of the slab.
- **side**: Which side to place the slab on. "topright", "top", and "right" are synonyms which cause the slab to be drawn on the top or the right depending on if orientation is "horizontal" or "vertical". "bottomleft", "bottom", and "left" are synonyms which cause the slab to be drawn on the bottom or the left depending on if orientation is "horizontal" or "vertical". "topleft" causes the slab to be drawn on the top or the left, and "bottomright" causes the slab to be drawn on the bottom or the right. "both" draws the slab mirrored on both sides (as in a violin plot).
- **scale**: What proportion of the region allocated to this geom to use to draw the slab. If scale = 1, slabs that use the maximum range will just touch each other. Default is 0.9 to leave some space.
- **justification**: Justification of the interval relative to the slab, where 0 indicates bottom/left justification and 1 indicates top/right justification (depending on orientation). If justification is NULL (the default), then it is set automatically based on the value of side: when side is "top"/"right" justification is set to 0, when side is "bottom"/"left" justification is set to 1, and when side is "both" justification is set to 0.5.
- **datatype**: When using composite geoms directly without a stat (e.g. `geom_slabinterval()`), datatype is used to indicate which part of the geom a row in the data targets: rows with datatype = "slab" target the slab portion of the geometry and rows with datatype = "interval" target the interval portion of the geometry. This is set automatically when using ggdist stats.

### Interval-specific aesthetics

- **xmin**: Left end of the interval sub-geometry (if orientation = "horizontal").
• **xmax**: Right end of the interval sub-geometry (if orientation = "horizontal").
• **ymin**: Lower end of the interval sub-geometry (if orientation = "vertical").
• **ymax**: Upper end of the interval sub-geometry (if orientation = "vertical").

**Point-specific aesthetics**

• **shape**: Shape type used to draw the **point** sub-geometry.

**Color aesthetics**

• **colour**: (or color) The color of the **interval** and **point** sub-geometries. Use the slab_color, interval_color, or point_color aesthetics (below) to set sub-geometry colors separately.
• **fill**: The fill color of the **slab** and **point** sub-geometries. Use the slab_fill or point_fill aesthetics (below) to set sub-geometry colors separately.
• **alpha**: The opacity of the **slab**, **interval**, and **point** sub-geometries. Use the slab_alpha, interval_alpha, or point_alpha aesthetics (below) to set sub-geometry colors separately.
• **colour_ramp**: (or color_ramp) A secondary scale that modifies the color scale to "ramp" to another color. See scale_colour_ramp() for examples.
• **fill_ramp**: A secondary scale that modifies the fill scale to "ramp" to another color. See scale_fill_ramp() for examples.

**Line aesthetics**

• **size**: Width of the outline around the **slab** (if visible). Also determines the width of the line used to draw the **interval** and the size of the **point**, but raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the slab_size, interval_size, or point_size aesthetics (below) to set sub-geometry line widths separately (note that when size is set directly using the override aesthetics, interval and point sizes are not affected by interval_size_domain, interval_size_range, and fatten_point).
• **stroke**: Width of the outline around the **point** sub-geometry.
• **linetype**: Type of line (e.g., "solid", "dashed", etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the slab_linetype or interval_linetype aesthetics (below) to set sub-geometry line types separately.

**Slab-specific color/line override aesthetics**

• **slab_fill**: Override for fill: the fill color of the slab.
• **slab_colour**: (or slab_color) Override for colour/color: the outline color of the slab.
• **slab_alpha**: Override for alpha: the opacity of the slab.
• **slab_size**: Override for size: the width of the outline of the slab.
• **slab_linetype**: Override for linetype: the line type of the outline of the slab.

**Interval-specific color/line override aesthetics**

• **interval_colour**: (or interval_color) Override for colour/color: the color of the interval.
stat_ccdfinterval

- **interval_alpha**: Override for alpha: the opacity of the interval.
- **interval_size**: Override for size: the line width of the interval.
- **interval_linetype**: Override for linetype: the line type of the interval.

**Point-specific color/line override aesthetics**

- **point_fill**: Override for fill: the fill color of the point.
- **point_colour**: (or *point_color*) Override for colour/color: the outline color of the point.
- **point_alpha**: Override for alpha: the opacity of the point.
- **point_size**: Override for size: the size of the point.

**Other aesthetics** (these work as in standard geoms)

- **width**
- **height**
- **group**

See examples of some of these aesthetics in action in vignette("slabinterval"). Learn more about the sub-geom override aesthetics (like *interval_color*) in the *scales* documentation. Learn more about basic *ggplot* aesthetics in vignette("ggplot2-specs").

**See Also**

See *geom_slabinterval()* for the geom underlying this stat. See *stat_slabinterval()* for the stat this shortcut is based on.

Other slabinterval stats: *stat_cdfinterval(), stat_eye(), stat_gradientinterval(), stat_halfeye(), stat_histinterval(), stat_interval(), stat_pointinterval(), stat_slab()*

**Examples**

```r
library(dplyr)
library(ggplot2)
library(distributional)

theme_set(theme_ggdist())

# ON SAMPLE DATA
set.seed(1234)
df = data.frame(
  group = c("a", "b", "c"),
  value = rnorm(1500, mean = c(5, 7, 9), sd = c(1, 1.5, 1))
)
df %>%
  ggplot(aes(x = value, y = group)) +
  stat_ccdfinterval() +
  expand_limits(x = 0)

# ON ANALYTICAL DISTRIBUTIONS
dist_df = data.frame(
```
group = c("a", "b", "c"),
mean = c( 5, 7, 8),
sd = c( 1, 1.5, 1)
)

# Vectorized distribution types, like distributional::dist_normal()
# and posterior::rvar(), can be used with the `xdist` / `ydist` aesthetics
dist_df %>%
  ggplot(aes(y = group, xdist = dist_normal(mean, sd))) +
  stat_cdfinterval() +
  expand_limits(x = 0)

---

**stat_cdfinterval**  
*CDF bar plot (shortcut stat)*

**Description**

Shortcut version of **stat_slabinterval()** with **geom_slabinterval()** for creating CDF bar plots.

Roughly equivalent to:

```r
stat_slabinterval(
  aes(thickness = stat(thickness(cdf)), justification = stat(0.5), side = stat("topleft"),
  slab_type = "cdf", normalize = "none", expand = TRUE
)
```

**Usage**

```r
stat_cdfinterval(
  mapping = NULL,
  data = NULL,
  geom = "slabinterval",
  position = "identity",
  ...
,
  slab_type = "cdf",
  normalize = "none",
  expand = TRUE,
  p_limits = c(NA, NA),
  adjust = 1,
  trim = TRUE,
  breaks = "Sturges",
  outline_bars = FALSE,
  point_interval = "median_qi",
  limits = NULL,
  n = 501,
  .width = c(0.66, 0.95),
  orientation = NA,
  na.rm = FALSE,
  show.legend = c(size = FALSE),
  inherit.aes = TRUE
)
```
Arguments

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

data The data to be displayed in this layer. There are three options:
- If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
- A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
- A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x, 10)`).

geom Use to override the default connection between `stat_cdfinterval()` and `geom_slabinterval()`.

position Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" (`position_dodge()`) or "dodgejust" (`position_dodgejust()`) can be useful if you have overlapping geometries.

... Other arguments passed to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3` (see Aesthetics, below). They may also be parameters to the paired geom/stat. When paired with the default geom, `geom_slabinterval()`, these include:

fill_type What type of fill to use when the fill color or alpha varies within a slab. One of:
- "segments": breaks up the slab geometry into segments for each unique combination of fill color and alpha value. This approach is supported by all graphics devices and works well for sharp cutoff values, but can give ugly results if a large number of unique fill colors are being used (as in gradients, like in `stat_gradientinterval()`).
- "gradient": a `grid::linearGradient()` is used to create a smooth gradient fill. This works well for large numbers of unique fill colors, but requires R >= 4.1 and is not yet supported on all graphics devices. As of this writing, the `png()` graphics device with `type = "cairo"`, the `svg()` device, the `pdf()` device, and the `ragg::agg_png()` devices are known to support this option. On R < 4.1, this option will fall back to `fill_type = "segment"` with a message.
- "auto": attempts to use `fill_type = "gradient"` if support for it can be auto-detected. On R >= 4.2, support for gradients can be auto-detected on some graphics devices; if support is not detected, this option will fall back to `fill_type = "segments"` (in case of a false negative, `fill_type = "gradient"` can be set explicitly). On R < 4.2, support for gradients cannot be auto-detected, so this will always fall back to `fill_type = "segments"`, in which case you can set `fill_type = "gradient"` explicitly if you are using a graphics device that support gradients.

interval_size_domain A length-2 numeric vector giving the minimum and maximum of the values of the size aesthetic that will be translated into...
actual sizes for intervals drawn according to interval_size_range (see the documentation for that argument.)

interval_size_range  A length-2 numeric vector. This geom scales the raw size aesthetic values when drawing interval and point sizes, as they tend to be too thick when using the default settings of `scale_size_continuous()`, which give sizes with a range of c(1, 6). The interval_size_domain value indicates the input domain of raw size values (typically this should be equal to the value of the range argument of the `scale_size_continuous()` function), and interval_size_range indicates the desired output range of the size values (the min and max of the actual sizes used to draw intervals). Most of the time it is not recommended to change the value of this argument, as it may result in strange scaling of legends; this argument is a holdover from earlier versions that did not have size aesthetics targeting the point and interval separately. If you want to adjust the size of the interval or points separately, you can instead use the interval_size or point_size aesthetics; see `scales`.

fatten_point  A multiplicative factor used to adjust the size of the point relative to the size of the thickest interval line. If you wish to specify point sizes directly, you can also use the point_size aesthetic and `scale_point_size_continuous()` or `scale_point_size_discrete()`; sizes specified with that aesthetic will not be adjusted using fatten_point.

slab_type  The type of slab function to calculate: probability density (or mass) function ("pdf"), cumulative distribution function ("cdf"), or complementary CDF ("ccdf").

normalize  How to normalize heights of functions input to the thickness aesthetic. One of:

- "all": normalize so that the maximum height across all data is 1.
- "panels": normalize within panels so that the maximum height in each panel is 1.
- "xy": normalize within the x/y axis opposite the orientation of this geom so that the maximum height at each value of the opposite axis is 1.
- "groups": normalize within values of the opposite axis and within each group so that the maximum height in each group is 1.
- "none": values are taken as is with no normalization (this should probably only be used with functions whose values are in [0,1], such as CDFs).

expand  For sample data, should the slab be expanded to the limits of the scale? Default FALSE. Can be length two to control expansion to the lower and upper limit respectively.

p_limits  Probability limits (as a vector of size 2) used to determine the lower and upper limits of the slab. E.g., if this is c(.001, .999), then a slab is drawn for the distribution from the quantile at p = .001 to the quantile at p = .999. If the lower (respectively upper) limit is NA, then the lower (upper) limit will be the minimum (maximum) of the distribution’s support if it is finite, and 0 (.001 (0.999) if it is not finite. E.g., if p_limits is c(NA, NA) on a gamma distribution the effective value of p_limits would be c(0, .999) since the gamma distribution is defined on (0, Inf); whereas on a normal distribution it would be equivalent to c(.001, .999) since the normal distribution is defined on (-Inf, Inf).
If slab_type is "pdf", bandwidth for the density estimator for sample data is adjusted by multiplying it by this value. See density() for more information.

For sample data, should the density estimate be trimmed to the range of the input data? Default TRUE.

If slab_type is "histogram", the breaks parameter that is passed to hist() to determine where to put breaks in the histogram (for sample data).

For sample data (if slab_type is "histogram") and for discrete analytical distributions (whose slabs are drawn as histograms), determines if outlines in between the bars are drawn when the slab_color aesthetic is used. If FALSE (the default), the outline is drawn only along the tops of the bars; if TRUE, outlines in between bars are also drawn.

A function from the point_interval() family (e.g., median_qi, mean_qi, mode_hdi, etc), or a string giving the name of a function from that family (e.g., "median_qi", "mean_qi", "mode_hdi", etc; if a string, the caller’s environment is searched for the function, followed by the ggdist environment). This function determines the point summary (typically mean, median, or mode) and interval type (quantile interval, qi; highest-density interval, hdi; or highest-density continuous interval, hdci). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of orientation. See the point_interval() family of functions for more information.

Manually-specified limits for the slab, as a vector of length two. These limits are combined with those computed based on p_limits as well as the limits defined by the scales of the plot to determine the limits used to draw the slab functions: these limits specify the maximal limits; i.e., if specified, the limits will not be wider than these (but may be narrower). Use NA to leave a limit alone; e.g. limits = c(0, NA) will ensure that the lower limit does not go below 0, but let the upper limit be determined by either p_limits or the scale settings.

Number of points at which to evaluate the function that defines the slab.

The .width argument passed to point_interval: a vector of probabilities to use that determine the widths of the resulting intervals. If multiple probabilities are provided, multiple intervals per group are generated, each with a different probability interval (and value of the corresponding .width and level generated variables).

Whether this geom is drawn horizontally or vertically. One of:

- NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
- "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify different groups. For each group, uses the x, xmin, xmax, and thickness aesthetics to draw points, intervals, and slabs.
- "vertical" (or "x"): draw vertically, using the x aesthetic to identify different groups. For each group, uses the y, ymin, ymax, and thickness aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for orientation, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal" (tidybayes had an orientation parameter before base ggplot did, hence the discrepancy).
If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

Should this layer be included in the legends? Default is `c(size = FALSE)`, unlike most geoms, to match its common use cases. FALSE hides all legends, TRUE shows all legends, and NA shows only those that are mapped (the default for most geoms).

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

Details

To visualize sample data, such as a data distribution, samples from a bootstrap distribution, or a Bayesian posterior, you can supply samples to the `x` or `y` aesthetic.

To visualize analytical distributions, you can use the `xdist` or `ydist` aesthetic. For historical reasons, you can also use `dist` to specify the distribution, though this is not recommended as it does not work as well with orientation detection. These aesthetics can be used as follows:

- `xdist`, `ydist`, and `dist` can be any distribution object from the distributional package (`dist_normal()`, `dist_beta()`, etc) or can be a `posterior::rvar()` object. Since these functions are vectorized, other columns can be passed directly to them in an `aes()` specification; e.g. `aes(dist = dist_normal(mu, sigma))` will work if `mu` and `sigma` are columns in the input data frame.

- `dist` can be a character vector giving the distribution name. Then the `arg1`, `arg2`, `arg9` aesthetics (or `args` as a list column) specify distribution arguments. Distribution names should correspond to R functions that have "p", "q", and "d" functions; e.g. "norm" is a valid distribution name because R defines the `pnorm()`, `qnorm()`, and `dnorm()` functions for Normal distributions.

See the `parse_dist()` function for a useful way to generate `dist` and `args` values from human-readable distribution specs (like "normal(0,1)"). Such specs are also produced by other packages (like the `brms::get_prior` function in brms); thus, `parse_dist()` combined with the stats described here can help you visualize the output of those functions.

Value

A `ggplot2::Stat` representing a CDF bar geometry which can be added to a `ggplot()` object.

Computed Variables

The following variables are computed by this stat and made available for use in aesthetic specifications (`aes()`) using the `stat()` or `after_stat()` functions:

- `x` or `y`: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is `x` or `y` depends on orientation
- `xmin` or `ymin`: For intervals, the lower end of the interval from the interval function.
- `xmax` or `ymax`: For intervals, the upper end of the interval from the interval function.
- `.width`: For intervals, the interval width as a numeric value in `[0, 1]`. For slabs, the width of the smallest interval containing that value of the slab.
Aesthetics

The slab+interval stats and geoms have a wide variety of aesthetics that control the appearance of their three sub-geometries: the slab, the point, and the interval.

These stats support the following aesthetics:

- **x**: x position of the geometry (when orientation = "vertical"); or sample data to be summarized (when orientation = "horizontal" with sample data).
- **y**: y position of the geometry (when orientation = "horizontal"); or sample data to be summarized (when orientation = "vertical" with sample data).
- **xdist**: When using analytical distributions, distribution to map on the x axis: a distributional object (e.g. `dist_normal()`) or a posterior::rvar() object.
- **ydist**: When using analytical distributions, distribution to map on the y axis: a distributional object (e.g. `dist_normal()`) or a posterior::rvar() object.
- **dist**: When using analytical distributions, a name of a distribution (e.g. "norm"), a distributional object (e.g. `dist_normal()`, or a posterior::rvar() object. See Details.
- **args**: Distribution arguments (args or arg1, ..., arg9). See Details.

In addition, in their default configuration (paired with geom_slabinterval()) the following aesthetics are supported by the underlying geom:

**Slab-specific aesthetics**

- **thickness**: The thickness of the slab at each x value (if orientation = "horizontal") or y value (if orientation = "vertical") of the slab.
- **side**: Which side to place the slab on. "topright", "top", and "right" are synonyms which cause the slab to be drawn on the top or the right depending on if orientation is "horizontal" or "vertical". "bottomleft", "bottom", and "left" are synonyms which cause the slab to be drawn on the bottom or the left depending on if orientation is "horizontal" or "vertical". "topleft" causes the slab to be drawn on the top or the left, and "bottomright" causes the slab to be drawn on the bottom or the right. "both" draws the slab mirrored on both sides (as in a violin plot).
• scale: What proportion of the region allocated to this geom to use to draw the slab. If scale = 1, slabs that use the maximum range will just touch each other. Default is 0.9 to leave some space.

• justification: Justification of the interval relative to the slab, where 0 indicates bottom/left justification and 1 indicates top/right justification (depending on orientation). If justification is NULL (the default), then it is set automatically based on the value of side: when side is "top"/"right" justification is set to 0, when side is "bottom"/"left" justification is set to 1, and when side is "both" justification is set to 0.5.

• datatype: When using composite geoms directly without a stat (e.g. `geom_slabinterval()`), datatype is used to indicate which part of the geom a row in the data targets: rows with datatype = "slab" target the slab portion of the geometry and rows with datatype = "interval" target the interval portion of the geometry. This is set automatically when using ggdist stats.

Interval-specific aesthetics

• xmin: Left end of the interval sub-geometry (if orientation = "horizontal").
• xmax: Right end of the interval sub-geometry (if orientation = "horizontal").
• ymin: Lower end of the interval sub-geometry (if orientation = "vertical").
• ymax: Upper end of the interval sub-geometry (if orientation = "vertical").

Point-specific aesthetics

• shape: Shape type used to draw the point sub-geometry.

Color aesthetics

• colour: (or color) The color of the interval and point sub-geometries. Use the slab_color, interval_color, or point_color aesthetics (below) to set sub-geometry colors separately.
• fill: The fill color of the slab and point sub-geometries. Use the slab_fill or point_fill aesthetics (below) to set sub-geometry colors separately.
• alpha: The opacity of the slab, interval, and point sub-geometries. Use the slab_alpha, interval_alpha, or point_alpha aesthetics (below) to set sub-geometry colors separately.
• colour_ramp: (or color_ramp) A secondary scale that modifies the color scale to "ramp" to another color. See `scale_colour_ramp()` for examples.
• fill_ramp: A secondary scale that modifies the fill scale to "ramp" to another color. See `scale_fill_ramp()` for examples.

Line aesthetics

• size: Width of the outline around the slab (if visible). Also determines the width of the line used to draw the interval and the size of the point, but raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the slab_size, interval_size, or point_size aesthetics (below) to set sub-geometry line widths separately (note that when size is set directly using the override aesthetics, interval and point sizes are not affected by interval_size_domain, interval_size_range, and fatten_point).
• stroke: Width of the outline around the point sub-geometry.
• `linetype`: Type of line (e.g., "solid", "dashed", etc) used to draw the `interval` and the outline of the `slab` (if it is visible). Use the `slab_linetype` or `interval_linetype` aesthetics (below) to set sub-geometry line types separately.

**Slab-specific color/line override aesthetics**

• `slab_fill`: Override for `fill`: the fill color of the slab.
• `slab_colour`: (or `slab_color`) Override for `colour/color`: the outline color of the slab.
• `slab_alpha`: Override for `alpha`: the opacity of the slab.
• `slab_size`: Override for `size`: the width of the outline of the slab.
• `slab_linetype`: Override for `linetype`: the line type of the outline of the slab.

**Interval-specific color/line override aesthetics**

• `interval_colour`: (or `interval_color`) Override for `colour/color`: the color of the interval.
• `interval_alpha`: Override for `alpha`: the opacity of the interval.
• `interval_size`: Override for `size`: the line width of the interval.
• `interval_linetype`: Override for `linetype`: the line type of the interval.

**Point-specific color/line override aesthetics**

• `point_fill`: Override for `fill`: the fill color of the point.
• `point_colour`: (or `point_color`) Override for `colour/color`: the outline color of the point.
• `point_alpha`: Override for `alpha`: the opacity of the point.
• `point_size`: Override for `size`: the size of the point.

**Other aesthetics** (these work as in standard geoms)

• `width`
• `height`
• `group`

See examples of some of these aesthetics in action in vignette("slabinterval"). Learn more about the sub-geom override aesthetics (like `interval_color`) in the scales documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").

**See Also**

See `geom_slabinterval()` for the geom underlying this stat. See `stat_slabinterval()` for the stat this shortcut is based on.

Other slabinterval stats: `stat_ccdfinterval()`, `stat_eye()`, `stat_gradientinterval()`, `stat_halfeye()`, `stat_histinterval()`, `stat_interval()`, `stat_pointinterval()`, `stat_slab()`
Examples

```r
library(dplyr)
library(ggplot2)
library(distributional)

theme_set(theme_ggdist())

# ON SAMPLE DATA
set.seed(1234)
df = data.frame(
  group = c("a", "b", "c"),
  value = rnorm(1500, mean = c(5, 7, 9), sd = c(1, 1.5, 1))
)
df %>%
  ggplot(aes(x = value, y = group)) +
  stat_cdfinterval()

# ON ANALYTICAL DISTRIBUTIONS
dist_df = data.frame(
  group = c("a", "b", "c"),
  mean = c( 5, 7, 8),
  sd = c( 1, 1.5, 1)
)
# Vectorized distribution types, like distributional::dist_normal()
# and posterior::rvar(), can be used with the `xdist` / `ydist` aesthetics
dist_df %>%
  ggplot(aes(y = group, xdist = dist_normal(mean, sd))) +
  stat_cdfinterval()
```

---

**stat_eye**

*Eye (violin + interval) plot (shortcut stat)*

**Description**

Shortcut version of `stat_slabinterval()` with `geom_slabinterval()` for creating eye (violin + interval) plots.

Roughly equivalent to:

```r
stat_slabinterval(
  aes(side = stat("both"))
)
```

**Usage**

```r
stat_eye(
  mapping = NULL,
  data = NULL,
  geom = "slabinterval",
```
position = "identity",
...,  
slab_type = "pdf",
p_limits = c(NA, NA),
adjust = 1,
trim = TRUE,
expand = FALSE,
breaks = "Sturges",
outline_bars = FALSE,
point_interval = "median_qi",
limits = NULL,
n = 501,
.width = c(0.66, 0.95),
orientation = NA,
na.rm = FALSE,
show.legend = c(size = FALSE),
inherit.aes = TRUE
)

Arguments

mapping Set of aesthetic mappings created by aes() or aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data The data to be displayed in this layer. There are three options:
If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot().
A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.
A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
geom Use to override the default connection between stat_eye() and geom_slabinterval()
position Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" (position_dodge()) or "dodgejust" (position_dodgejust()) can be useful if you have overlapping geometries.
...

Other arguments passed to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3 (see Aesthetics, below). They may also be parameters to the paired geom/stat. When paired with the default geom, geom_slabinterval(), these include:
normalize How to normalize heights of functions input to the thickness aesthetic. One of:
  • "all": normalize so that the maximum height across all data is 1.
  • "panels": normalize within panels so that the maximum height in each panel is 1.
• "xy": normalize within the x/y axis opposite the orientation of this geom so that the maximum height at each value of the opposite axis is 1.

• "groups": normalize within values of the opposite axis and within each group so that the maximum height in each group is 1.

• "none": values are taken as is with no normalization (this should probably only be used with functions whose values are in [0,1], such as CDFs).

fill_type What type of fill to use when the fill color or alpha varies within a slab. One of:

• "segments": breaks up the slab geometry into segments for each unique combination of fill color and alpha value. This approach is supported by all graphics devices and works well for sharp cutoff values, but can give ugly results if a large number of unique fill colors are being used (as in gradients, like in stat_gradientinterval()).

• "gradient": a grid::linearGradient() is used to create a smooth gradient fill. This works well for large numbers of unique fill colors, but requires R >= 4.1 and is not yet supported on all graphics devices. As of this writing, the png() graphics device with type = "cairo", the svg() device, the pdf() device, and the ragg::agg_png() devices are known to support this option. On R < 4.1, this option will fall back to fill_type = "segment" with a message.

• "auto": attempts to use fill_type = "gradient" if support for it can be auto-detected. On R >= 4.2, support for gradients can be auto-detected on some graphics devices; if support is not detected, this option will fall back to fill_type = "segments" (in case of a false negative, fill_type = "gradient" can be set explicitly). On R < 4.2, support for gradients cannot be auto-detected, so this will always fall back to fill_type = "segments", in which case you can set fill_type = "gradient" explicitly if you are using a graphics device that supports gradients.

interval_size_domain A length-2 numeric vector giving the minimum and maximum of the values of the size aesthetic that will be translated into actual sizes for intervals drawn according to interval_size_range (see the documentation for that argument.)

interval_size_range A length-2 numeric vector. This geom scales the raw size aesthetic values when drawing interval and point sizes, as they tend to be too thick when using the default settings of scale_size_continuous(), which give sizes with a range of c(1, 6). The interval_size_domain value indicates the input domain of raw size values (typically this should be equal to the value of the range argument of the scale_size_continuous() function), and interval_size_range indicates the desired output range of the size values (the min and max of the actual sizes used to draw intervals). Most of the time it is not recommended to change the value of this argument, as it may result in strange scaling of legends; this argument is a holdover from earlier versions that did not have size aesthetics targeting the point and interval separately. If you want to adjust the size of the interval or
points separately, you can instead use the `interval_size` or `point_size` aesthetics; see `scales`.

`fatten_point` A multiplicative factor used to adjust the size of the point relative to the size of the thickest interval line. If you wish to specify point sizes directly, you can also use the `point_size` aesthetic and `scale_point_size_continuous()` or `scale_point_size_discrete()`: sizes specified with that aesthetic will not be adjusted using `fatten_point`.

`slab_type` The type of slab function to calculate: probability density (or mass) function ("pdf"), cumulative distribution function ("cdf"), or complementary CDF ("ccdf").

`p_limits` Probability limits (as a vector of size 2) used to determine the lower and upper limits of the slab. E.g., if this is `c(.001, .999)`, then a slab is drawn for the distribution from the quantile at $p = .001$ to the quantile at $p = .999$. If the lower (respectively upper) limit is NA, then the lower (upper) limit will be the minimum (maximum) of the distribution’s support if it is finite, and 0 .001 (0 .999) if it is not finite. E.g., if `p_limits` is `c(NA, NA)` on a gamma distribution the effective value of `p_limits` would be `c(0, .999)` since the gamma distribution is defined on (0, Inf); whereas on a normal distribution it would be equivalent to `c(.001, .999)` since the normal distribution is defined on (-Inf, Inf).

`adjust` If `slab_type` is "pdf", bandwidth for the density estimator for sample data is adjusted by multiplying it by this value. See `density()` for more information.

`trim` For sample data, should the density estimate be trimmed to the range of the input data? Default TRUE.

`expand` For sample data, should the slab be expanded to the limits of the scale? Default FALSE. Can be length two to control expansion to the lower and upper limit respectively.

`breaks` If `slab_type` is "histogram", the breaks parameter that is passed to `hist()` to determine where to put breaks in the histogram (for sample data).

`outline_bars` For sample data (if `slab_type` is "histogram") and for discrete analytical distributions (whose slabs are drawn as histograms), determines if outlines in between the bars are drawn when the `slab_color` aesthetic is used. If FALSE (the default), the outline is drawn only along the tops of the bars; if TRUE, outlines in between bars are also drawn.

`point_interval` A function from the `point_interval()` family (e.g., `median_qi`, `mean_qi`, `mode_hdi`, etc), or a string giving the name of a function from that family (e.g., "median_qi", "mean_qi", "mode_hdi", etc; if a string, the caller’s environment is searched for the function, followed by the `ggdist` environment). This function determines the point summary (typically mean, median, or mode) and interval type (quantile interval, qi; highest-density interval, hdi; or highest-density continuous interval, hdci). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of `orientation`. See the `point_interval()` family of functions for more information.

`limits` Manually-specified limits for the slab, as a vector of length two. These limits are combined with those computed based on `p_limits` as well as the limits defined by the scales of the plot to determine the limits used to draw the slab functions: these limits specify the maximal limits; i.e., if specified, the limits will not be wider than these (but may be narrower). Use NA to leave a limit alone; e.g.
limits = c(0, NA) will ensure that the lower limit does not go below 0, but let
the upper limit be determined by either p_limits or the scale settings.

n  Number of points at which to evaluate the function that defines the slab.

.width  The .width argument passed to point_interval: a vector of probabilities to
use that determine the widths of the resulting intervals. If multiple probabilities
are provided, multiple intervals per group are generated, each with a different
probability interval (and value of the corresponding .width and level gener-
ated variables).

orientation  Whether this geom is drawn horizontally or vertically. One of:
  • NA (default): automatically detect the orientation based on how the aesthet-
    ics are assigned. Automatic detection works most of the time.
  • "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify
different groups. For each group, uses the x, xmin, xmax, and thickness
aesthetics to draw points, intervals, and slabs.
  • "vertical" (or "x"): draw vertically, using the x aesthetic to identify dif-
    ferent groups. For each group, uses the y, ymin, ymax, and thickness
aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for orientation, "x"
can be used as an alias for "vertical" and "y" as an alias for "horizontal"
tidybayes had an orientation parameter before base ggplot did, hence the
discrepancy).

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

show.legend  Should this layer be included in the legends? Default is c(size = FALSE), unlike
most geoms, to match its common use cases. FALSE hides all legends, TRUE
shows all legends, and NA shows only those that are mapped (the default for
most geoms).

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

Details

To visualize sample data, such as a data distribution, samples from a bootstrap distribution, or a
Bayesian posterior, you can supply samples to the x or y aesthetic.

To visualize analytical distributions, you can use the xdist or ydist aesthetic. For historical
reasons, you can also use dist to specify the distribution, though this is not recommended as it
does not work as well with orientation detection. These aesthetics can be used as follows:

• xdist, ydist, and dist can be any distribution object from the distributional package (dist_normal(),
dist_beta(), etc) or can be a posterior:rvar() object. Since these functions are vector-
ized, other columns can be passed directly to them in an aes() specification; e.g. aes(dist =
dist_normal(mu, sigma)) will work if mu and sigma are columns in the input data frame.

• dist can be a character vector giving the distribution name. Then the arg1, ... arg9 aesthet-
ics (or args as a list column) specify distribution arguments. Distribution names should
correspond to R functions that have "p", "q", and "d" functions; e.g. "norm" is a valid distribution name because R defines the \texttt{pnorm()}, \texttt{qnorm()}, and \texttt{dnorm()} functions for Normal distributions.

See the \texttt{parse_dist()} function for a useful way to generate dist and args values from human-readable distribution specs (like "normal(0,1)"). Such specs are also produced by other packages (like the \texttt{brms::get_prior} function in brms); thus, \texttt{parse_dist()} combined with the stats described here can help you visualize the output of those functions.

**Value**

A \texttt{ggplot2::Stat} representing a eye (violin + interval) geometry which can be added to a \texttt{ggplot()} object.

**Computed Variables**

The following variables are computed by this stat and made available for use in aesthetic specifications (\texttt{aes()}) using the \texttt{stat()} or \texttt{after_stat()} functions:

- \texttt{x} or \texttt{y}: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is \texttt{x} or \texttt{y} depends on orientation.
- \texttt{xmin} or \texttt{ymin}: For intervals, the lower end of the interval from the interval function.
- \texttt{xmax} or \texttt{ymax}: For intervals, the upper end of the interval from the interval function.
- \texttt{.width}: For intervals, the interval width as a numeric value in \([0, 1]\). For slabs, the width of the smallest interval containing that value of the slab.
- \texttt{level}: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.
- \texttt{pdf}: For slabs, the probability density function (PDF). If \texttt{options("ggdist.experimental.slab_data_in_intervals")} is \texttt{TRUE}: For intervals, the PDF at the point summary; intervals also have \texttt{pdf_min} and \texttt{pdf_max} for the PDF at the lower and upper ends of the interval.
- \texttt{cdf}: For slabs, the cumulative distribution function. If \texttt{options("ggdist.experimental.slab_data_in_intervals")} is \texttt{TRUE}: For intervals, the CDF at the point summary; intervals also have \texttt{cdf_min} and \texttt{cdf_max} for the CDF at the lower and upper ends of the interval.
- \texttt{f}: For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by \texttt{slab_type}.
- \texttt{n}: For slabs, the number of data points summarized into that slab. If the slab was created from an analytical distribution via the \texttt{xdist}, \texttt{ydist}, or \texttt{dist} aesthetic, \texttt{n} will be \texttt{Inf}.

**Aesthetics**

The slab+interval stats and geoms have a wide variety of aesthetics that control the appearance of their three sub-geometries: the \texttt{slab}, the \texttt{point}, and the \texttt{interval}.

These stats support the following aesthetics:

- \texttt{x}: \texttt{x} position of the geometry (when \texttt{orientation = "vertical"}); or sample data to be summarized (when \texttt{orientation = "horizontal"} with sample data).
- \texttt{y}: \texttt{y} position of the geometry (when \texttt{orientation = "horizontal"}); or sample data to be summarized (when \texttt{orientation = "vertical"} with sample data).
• xdist: When using analytical distributions, distribution to map on the x axis: a distributional object (e.g. `dist_normal()`) or a posterior::rvar() object.

• ydist: When using analytical distributions, distribution to map on the y axis: a distributional object (e.g. `dist_normal()`) or a posterior::rvar() object.

• dist: When using analytical distributions, a name of a distribution (e.g. "norm"), a distributional object (e.g. `dist_normal()`), or a posterior::rvar() object. See Details.

• args: Distribution arguments (args or arg1, ... arg9). See Details.

In addition, in their default configuration (paired with `geom_slabinterval()`) the following aesthetics are supported by the underlying geom:

**Slab-specific aesthetics**

• thickness: The thickness of the slab at each x value (if orientation = "horizontal") or y value (if orientation = "vertical") of the slab.

• side: Which side to place the slab on. "topright", "top", and "right" are synonyms which cause the slab to be drawn on the top or the right depending on if orientation is "horizontal" or "vertical". "bottomleft", "bottom", and "left" are synonyms which cause the slab to be drawn on the bottom or the left depending on if orientation is "horizontal" or "vertical". "topleft" causes the slab to be drawn on the top or the left, and "bottomright" causes the slab to be drawn on the bottom or the right. "both" draws the slab mirrored on both sides (as in a violin plot).

• scale: What proportion of the region allocated to this geom to use to draw the slab. If scale = 1, slabs that use the maximum range will just touch each other. Default is 0.9 to leave some space.

• justification: Justification of the interval relative to the slab, where 0 indicates bottom/left justification and 1 indicates top/right justification (depending on orientation). If justification is NULL (the default), then it is set automatically based on the value of side: when side is "top"/"right" justification is set to 0, when side is "bottom"/"left" justification is set to 1, and when side is "both" justification is set to 0.5.

• datatype: When using composite geoms directly without a stat (e.g. `geom_slabinterval()`), datatype is used to indicate which part of the geom a row in the data targets: rows with datatype = "slab" target the slab portion of the geometry and rows with datatype = "interval" target the interval portion of the geometry. This is set automatically when using ggdist stats.

**Interval-specific aesthetics**

• xmin: Left end of the interval sub-geometry (if orientation = "horizontal").

• xmax: Right end of the interval sub-geometry (if orientation = "horizontal").

• ymin: Lower end of the interval sub-geometry (if orientation = "vertical").

• ymax: Upper end of the interval sub-geometry (if orientation = "vertical").

**Point-specific aesthetics**

• shape: Shape type used to draw the point sub-geometry.

**Color aesthetics**
• colour: (or color) The color of the interval and point sub-geometries. Use the slab_color, interval_color, or point_color aesthetics (below) to set sub-geometry colors separately.

• fill: The fill color of the slab and point sub-geometries. Use the slab_fill or point_fill aesthetics (below) to set sub-geometry colors separately.

• alpha: The opacity of the slab, interval, and point sub-geometries. Use the slab_alpha, interval_alpha, or point_alpha aesthetics (below) to set sub-geometry colors separately.

• colour_ramp: (or color_ramp) A secondary scale that modifies the color scale to "ramp" to another color. See scale_colour_ramp() for examples.

• fill_ramp: A secondary scale that modifies the fill scale to "ramp" to another color. See scale_fill_ramp() for examples.

Line aesthetics

• size: Width of the outline around the slab (if visible). Also determines the width of the line used to draw the interval and the size of the point, but raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the slab_size, interval_size, or point_size aesthetics (below) to set sub-geometry line widths separately (note that when size is set directly using the override aesthetics, interval and point sizes are not affected by interval_size_domain, interval_size_range, and fatten_point).

• stroke: Width of the outline around the point sub-geometry.

• linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the interval and the outline of the slab (if it is visible). Use the slab_linetype or interval_linetype aesthetics (below) to set sub-geometry line types separately.

Slab-specific color/line override aesthetics

• slab_fill: Override for fill: the fill color of the slab.

• slab_colour: (or slab_color) Override for colour/color: the outline color of the slab.

• slab_alpha: Override for alpha: the opacity of the slab.

• slab_size: Override for size: the width of the outline of the slab.

• slab_linetype: Override for linetype: the line type of the outline of the slab.

Interval-specific color/line override aesthetics

• interval_colour: (or interval_color) Override for colour/color: the color of the interval.

• interval_alpha: Override for alpha: the opacity of the interval.

• interval_size: Override for size: the line width of the interval.

• interval_linetype: Override for linetype: the line type of the interval.

Point-specific color/line override aesthetics

• point_fill: Override for fill: the fill color of the point.

• point_colour: (or point_color) Override for colour/color: the outline color of the point.
• point_alpha: Override for alpha: the opacity of the point.
• point_size: Override for size: the size of the point.

Other aesthetics (these work as in standard geoms)

• width
• height
• group

See examples of some of these aesthetics in action in vignette("slabinterval"). Learn more about the sub-geom override aesthetics (like interval_color) in the scales documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").

See Also

See geom_slabinterval() for the geom underlying this stat. See stat_slabinterval() for the stat this shortcut is based on.

Other slabinterval stats: stat_ccdfinterval(), stat_cdfinterval(), stat_gradientinterval(), stat_halfeye(), stat_histinterval(), stat_interval(), stat_pointinterval(), stat_slab()

Examples

library(dplyr)
library(ggplot2)
library(distributional)

theme_set(theme_ggdist())

# ON SAMPLE DATA
set.seed(1234)
df = data.frame(
  group = c("a", "b", "c"),
  value = rnorm(1500, mean = c(5, 7, 9), sd = c(1, 1.5, 1))
)
df %>%
ggplot(aes(x = value, y = group)) +
stat_eye()

# ON ANALYTICAL DISTRIBUTIONS
dist_df = data.frame(
  group = c("a", "b", "c"),
  mean = c( 5, 7, 8),
  sd = c( 1, 1.5, 1)
)
# Vectorized distribution types, like distributional::dist_normal()
# and posterior::rvar(), can be used with the 'xdist' / 'ydist' aesthetics
dist_df %>%
ggplot(aes(y = group, xdist = dist_normal(mean, sd))) +
stat_eye()
stat_gradientinterval  Gradient + interval plot (shortcut stat)

Description

Shortcut version of stat_slabinterval() with geom_slabinterval() for creating gradient + interval plots.

Roughly equivalent to:

```r
stat_slabinterval(
    aes(justification = stat(0.5), thickness = stat(thickness(1)), slab_alpha = stat(f)),
    fill_type = "auto",
    show.legend = c(size = FALSE, slab_alpha = FALSE)
)
```

If your graphics device supports it, it is recommended to use this stat with fill_type = "gradient" (see the description of that parameter). On R >= 4.2, support for fill_type = "gradient" should be auto-detected based on the graphics device you are using.

Usage

```r
stat_gradientinterval(
    mapping = NULL,
    data = NULL,
    geom = "slabinterval",
    position = "identity",
    ..., 
    fill_type = "auto",
    slab_type = "pdf",
    p_limits = c(NA, NA),
    adjust = 1,
    trim = TRUE,
    expand = FALSE,
    breaks = "Sturges",
    outline_bars = FALSE,
    point_interval = "median_qi",
    limits = NULL,
    n = 501,
    .width = c(0.66, 0.95),
    orientation = NA,
    na.rm = FALSE,
    show.legend = c(size = FALSE, slab_alpha = FALSE),
    inherit.aes = TRUE
)
```
Arguments

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

data The data to be displayed in this layer. There are three options:
- If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
- A data frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
- A function will be called with a single argument, the plot data. The return value must be a data frame, and will be used as the layer data. A function can be created from a formula (e.g., `~ head(x, 10)`).

geom Use to override the default connection between `stat_gradientinterval()` and `geom_slabinterval()`

position Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" (position_dodge()) or "dodgejust" (position_dodgejust()) can be useful if you have overlapping geometries.

... Other arguments passed to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3` (see `Aesthetics`, below). They may also be parameters to the paired geom/stat. When paired with the default geom, `geom_slabinterval()`, these include:

- normalize How to normalize heights of functions input to the thickness aesthetic. One of:
  - "all": normalize so that the maximum height across all data is 1.
  - "panels": normalize within panels so that the maximum height in each panel is 1.
  - "xy": normalize within the x/y axis opposite the orientation of this geom so that the maximum height at each value of the opposite axis is 1.
  - "groups": normalize within values of the opposite axis and within each group so that the maximum height in each group is 1.
  - "none": values are taken as is with no normalization (this should probably only be used with functions whose values are in [0,1], such as CDFs).

- interval_size_domain A length-2 numeric vector giving the minimum and maximum of the values of the size aesthetic that will be translated into actual sizes for intervals drawn according to `interval_size_range` (see the documentation for that argument.)

- interval_size_range A length-2 numeric vector. This geom scales the raw size aesthetic values when drawing interval and point sizes, as they tend to be too thick when using the default settings of `scale_size_continuous()`, which give sizes with a range of c(1, 6). The `interval_size_domain` value indicates the input domain of raw size values (typically this should be equal to the value of the range argument of the `scale_size_continuous()` function).
function), and interval_size_range indicates the desired output range of
the size values (the min and max of the actual sizes used to draw inter-
vals). Most of the time it is not recommended to change the value of this
argument, as it may result in strange scaling of legends; this argument is a
holdover from earlier versions that did not have size aesthetics targeting the
point and interval separately. If you want to adjust the size of the interval or
points separately, you can instead use the interval_size or point_size
aesthetics; see scales.

fatten_point  A multiplicative factor used to adjust the size of the point relat-
ive to the size of the thickest interval line. If you wish to specify point sizes
directly, you can also use the point_size aesthetic and scale_point_size_continuous()
or scale_point_size_discrete(); sizes specified with that aesthetic will
not be adjusted using fatten_point.

fill_type  What type of fill to use when the fill color or alpha varies within a slab. One of:

- "segments": breaks up the slab geometry into segments for each unique
  combination of fill color and alpha value. This approach is supported by all
  graphics devices and works well for sharp cutoff values, but can give ugly
  results if a large number of unique fill colors are being used (as in gradients,
  like in stat_gradientinterval()).
- "gradient": a grid::linearGradient() is used to create a smooth gra-
dient fill. This works well for large numbers of unique fill colors, but re-
quires R >= 4.1 and is not yet supported on all graphics devices. As of
this writing, the png() graphics device with type = "cairo", the svg() device, the pdf() device, and the ragg::agg_png() devices are known to
support this option. On R < 4.1, this option will fall back to fill_type = "segment"
with a message.
- "auto": attempts to use fill_type = "gradient" if support for it can be
  auto-detected. On R >= 4.2, support for gradients can be auto-detected
on some graphics devices; if support is not detected, this option will fall
back to fill_type = "segments" (in case of a false negative, fill_type = "gradient" can be set explicitly). On R < 4.2, support for gradients cannot
be auto-detected, so this will always fall back to fill_type = "segments",
in which case you can set fill_type = "gradient" explicitly if you are
using a graphics device that support gradients.

slab_type  The type of slab function to calculate: probability density (or mass) function
("pdf"), cumulative distribution function ("cdf"), or complementary CDF ("ccdf").

p_limits  Probability limits (as a vector of size 2) used to determine the lower and upper
limits of the slab. E.g., if this is c(.001, .999), then a slab is drawn for the dis-
tribution from the quantile at p = .001 to the quantile at p = .999. If the lower
(respectively upper) limit is NA, then the lower (upper) limit will be the mini-
mum (maximum) of the distribution’s support if it is finite, and 0 .001 (0.999)
if it is not finite. E.g., if p_limits is c(NA, NA) on a gamma distribution the
effective value of p_limits would be c(0, .999) since the gamma distribution
is defined on (0, Inf); whereas on a normal distribution it would be equivalent
to c(.001, .999) since the normal distribution is defined on (-Inf, Inf).

adjust  If slab_type is "pdf", bandwidth for the density estimator for sample data is
adjusted by multiplying it by this value. See density() for more information.
trim
For sample data, should the density estimate be trimmed to the range of the input data? Default TRUE.

expand
For sample data, should the slab be expanded to the limits of the scale? Default FALSE. Can be length two to control expansion to the lower and upper limit respectively.

breaks
If slab_type is "histogram", the breaks parameter that is passed to hist() to determine where to put breaks in the histogram (for sample data).

outline_bars
For sample data (if slab_type is "histogram") and for discrete analytical distributions (whose slabs are drawn as histograms), determines if outlines in between the bars are drawn when the slab_color aesthetic is used. If FALSE (the default), the outline is drawn only along the tops of the bars; if TRUE, outlines in between bars are also drawn.

point_interval
A function from the point_interval() family (e.g., median_qi, mean_qi, mode_hdi, etc), or a string giving the name of a function from that family (e.g., "median_qi", "mean_qi", "mode_hdi", etc; if a string, the caller’s environment is searched for the function, followed by the ggdist environment). This function determines the point summary (typically mean, median, or mode) and interval type (quantile interval, qi; highest-density interval, hdi; or highest-density continuous interval, hdci). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of orientation. See the point_interval() family of functions for more information.

limits
Manually-specified limits for the slab, as a vector of length two. These limits are combined with those computed based on p_limits as well as the limits defined by the scales of the plot to determine the limits used to draw the slab functions: these limits specify the maximal limits; i.e., if specified, the limits will not be wider than these (but may be narrower). Use NA to leave a limit alone; e.g. limits = c(0, NA) will ensure that the lower limit does not go below 0, but let the upper limit be determined by either p_limits or the scale settings.

n
Number of points at which to evaluate the function that defines the slab.

.width
The .width argument passed to point_interval: a vector of probabilities to use that determine the widths of the resulting intervals. If multiple probabilities are provided, multiple intervals per group are generated, each with a different probability interval (and value of the corresponding .width and level generated variables).

orientation
Whether this geom is drawn horizontally or vertically. One of:

- NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
- "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify different groups. For each group, uses the x, xmin, xmax, and thickness aesthetics to draw points, intervals, and slabs.
- "vertical" (or "x"): draw vertically, using the x aesthetic to identify different groups. For each group, uses the y, ymin, ymax, and thickness aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for orientation, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal".
(tidybayes had an orientation parameter before base ggplot did, hence the discrepancy).

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

show.legend
Should this layer be included in the legends? Default is c(size = FALSE), unlike most geoms, to match its common use cases. FALSE hides all legends, TRUE shows all legends, and NA shows only those that are mapped (the default for most geoms).

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

Details
To visualize sample data, such as a data distribution, samples from a bootstrap distribution, or a Bayesian posterior, you can supply samples to the x or y aesthetic.

To visualize analytical distributions, you can use the xdist or ydist aesthetic. For historical reasons, you can also use dist to specify the distribution, though this is not recommended as it does not work as well with orientation detection. These aesthetics can be used as follows:

• xdist, ydist, and dist can be any distribution object from the distributional package (dist_normal(), dist_beta(), etc) or can be a posterior::rvar() object. Since these functions are vectorized, other columns can be passed directly to them in an aes() specification; e.g. aes(dist = dist_normal(mu, sigma)) will work if mu and sigma are columns in the input data frame.
• dist can be a character vector giving the distribution name. Then the arg1, ... arg9 aesthetics (or args as a list column) specify distribution arguments. Distribution names should correspond to R functions that have "p", "q", and "d" functions; e.g. "norm" is a valid distribution name because R defines the pnorm(), qnorm(), and dnorm() functions for Normal distributions.

See the parse_dist() function for a useful way to generate dist and args values from human-readable distribution specs (like “normal(0,1)”). Such specs are also produced by other packages (like the brms::get_prior function in brms); thus, parse_dist() combined with the stats described here can help you visualize the output of those functions.

Value
A ggplot2::Stat representing a gradient + interval geometry which can be added to a ggplot() object.

Computed Variables
The following variables are computed by this stat and made available for use in aesthetic specifications (aes()) using the stat() or after_stat() functions:

• x or y: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is x or y depends on orientation
• xmin or ymin: For intervals, the lower end of the interval from the interval function.
• xmax or ymax: For intervals, the upper end of the interval from the interval function.
• `.width`: For intervals, the interval width as a numeric value in \([0, 1]\). For slabs, the width of the smallest interval containing that value of the slab.

• `.level`: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.

• `pdf`: For slabs, the probability density function (PDF). If `options("ggdist.experimental.slab_data_in_intervals")` is `TRUE`: For intervals, the PDF at the point summary; intervals also have `pdf_min` and `pdf_max` for the PDF at the lower and upper ends of the interval.

• `cdf`: For slabs, the cumulative distribution function. If `options("ggdist.experimental.slab_data_in_intervals")` is `TRUE`: For intervals, the CDF at the point summary; intervals also have `cdf_min` and `cdf_max` for the CDF at the lower and upper ends of the interval.

• `f`: For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by `slab_type`.

• `n`: For slabs, the number of data points summarized into that slab. If the slab was created from an analytical distribution via the `xdist`, `ydist`, or `dist` aesthetic, `n` will be `Inf`.

Aesthetics

The slab+interval stats and geoms have a wide variety of aesthetics that control the appearance of their three sub-geometries: the `slab`, the `point`, and the `interval`.

These stats support the following aesthetics:

• `x`: x position of the geometry (when orientation = "vertical"); or sample data to be summarized (when orientation = "horizontal" with sample data).

• `y`: y position of the geometry (when orientation = "horizontal"); or sample data to be summarized (when orientation = "vertical" with sample data).

• `xdist`: When using analytical distributions, distribution to map on the x axis: a `distributional` object (e.g. `dist_normal()`) or a `posterior::rvar()` object.

• `ydist`: When using analytical distributions, distribution to map on the y axis: a `distributional` object (e.g. `dist_normal()`) or a `posterior::rvar()` object.

• `dist`: When using analytical distributions, a name of a distribution (e.g. "norm"), a `distributional` object (e.g. `dist_normal()`), or a `posterior::rvar()` object. See Details.

• `args`: Distribution arguments (`args` or `arg1, ... arg9`). See Details.

In addition, in their default configuration (paired with `geom_slabinterval()`) the following aesthetics are supported by the underlying geom:

Slab-specific aesthetics

• `thickness`: The thickness of the slab at each x value (if orientation = "horizontal") or y value (if orientation = "vertical") of the slab.

• `side`: Which side to place the slab on. "topright", "top", and "right" are synonyms which cause the slab to be drawn on the top or the right depending on if orientation is "horizontal" or "vertical". "bottomleft", "bottom", and "left" are synonyms which cause the slab to be drawn on the bottom or the left depending on if orientation is "horizontal" or "vertical". "topleft" causes the slab to be drawn on the top or the left, and "bottomright" causes the slab to be drawn on the bottom or the right. "both" draws the slab mirrored on both sides (as in a violin plot).
• **scale**: What proportion of the region allocated to this geom to use to draw the slab. If scale = 1, slabs that use the maximum range will just touch each other. Default is 0.9 to leave some space.

• **justification**: Justification of the interval relative to the slab, where 0 indicates bottom/left justification and 1 indicates top/right justification (depending on orientation). If justification is NULL (the default), then it is set automatically based on the value of side: when side is "top"/"right" justification is set to 0, when side is "bottom"/"left" justification is set to 1, and when side is "both" justification is set to 0.5.

• **datatype**: When using composite geoms directly without a stat (e.g. `geom_slabinterval()`), datatype is used to indicate which part of the geom a row in the data targets: rows with datatype = "slab" target the slab portion of the geometry and rows with datatype = "interval" target the interval portion of the geometry. This is set automatically when using ggdist stats.

**Interval-specific aesthetics**

• **xmin**: Left end of the interval sub-geometry (if orientation = "horizontal").

• **xmax**: Right end of the interval sub-geometry (if orientation = "horizontal").

• **ymin**: Lower end of the interval sub-geometry (if orientation = "vertical").

• **ymax**: Upper end of the interval sub-geometry (if orientation = "vertical").

**Point-specific aesthetics**

• **shape**: Shape type used to draw the point sub-geometry.

**Color aesthetics**

• **colour**: (or color) The color of the interval and point sub-geometries. Use the slab_color, interval_color, or point_color aesthetics (below) to set sub-geometry colors separately.

• **fill**: The fill color of the slab and point sub-geometries. Use the slab_fill or point_fill aesthetics (below) to set sub-geometry colors separately.

• **alpha**: The opacity of the slab, interval, and point sub-geometries. Use the slab_alpha, interval_alpha, or point_alpha aesthetics (below) to set sub-geometry colors separately.

• **colour_ramp**: (or color_ramp) A secondary scale that modifies the color scale to "ramp" to another color. See `scale_colour_ramp()` for examples.

• **fill_ramp**: A secondary scale that modifies the fill scale to "ramp" to another color. See `scale_fill_ramp()` for examples.

**Line aesthetics**

• **size**: Width of the outline around the slab (if visible). Also determines the width of the line used to draw the interval and the size of the point, but raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the slab_size, interval_size, or point_size aesthetics (below) to set sub-geometry line widths separately (note that when size is set directly using the override aesthetics, interval and point sizes are not affected by interval_size_domain, interval_size_range, and fatten_point).

• **stroke**: Width of the outline around the point sub-geometry.
• linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the interval and the outline of the slab (if it is visible). Use the slab_linetype or interval_linetype aesthetics (below) to set sub-geometry line types separately.

Slab-specific color/line override aesthetics

• slab_fill: Override for fill: the fill color of the slab.
• slab_colour: (or slab_color) Override for colour/color: the outline color of the slab.
• slab_alpha: Override for alpha: the opacity of the slab.
• slab_size: Override for size: the width of the outline of the slab.
• slab_linetype: Override for linetype: the line type of the outline of the slab.

Interval-specific color/line override aesthetics

• interval_colour: (or interval_color) Override for colour/color: the color of the interval.
• interval_alpha: Override for alpha: the opacity of the interval.
• interval_size: Override for size: the line width of the interval.
• interval_linetype: Override for linetype: the line type of the interval.

Point-specific color/line override aesthetics

• point_fill: Override for fill: the fill color of the point.
• point_colour: (or point_color) Override for colour/color: the outline color of the point.
• point_alpha: Override for alpha: the opacity of the point.
• point_size: Override for size: the size of the point.

Other aesthetics (these work as in standard geoms)

• width
• height
• group

See examples of some of these aesthetics in action in vignette("slabinterval"). Learn more about the sub-geom override aesthetics (like interval_color) in the scales documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").

See Also

See geom_slabinterval() for the geom underlying this stat. See stat_slabinterval() for the stat this shortcut is based on.

Other slabinterval stats: stat_ccdfinterval(), stat_cdfinterval(), stat_eye(), stat_halfeye(), stat_histinterval(), stat_interval(), stat_pointinterval(), stat_slab()
Examples

library(dplyr)
library(ggplot2)
library(distributional)

tHEME_set(THEME_ggdist())

# ON SAMPLE DATA
set.seed(1234)
df = data.frame(
  group = c("a", "b", "c"),
  value = rnorm(1500, mean = c(5, 7, 9), sd = c(1, 1.5, 1))
)
df %>%
  ggplot(aes(x = value, y = group)) +
  stat_gradientinterval()

# ON ANALYTICAL DISTRIBUTIONS
dist_df = data.frame(
  group = c("a", "b", "c"),
  mean = c( 5, 7, 8),
  sd = c( 1, 1.5, 1)
)
# Vectorized distribution types, like distributional::dist_normal()
# and posterior::rvar(), can be used with the `xdist` / `ydist` aesthetics
dist_df %>%
  ggplot(aes(y = group, xdist = dist_normal(mean, sd))) +
  stat_gradientinterval()

stat_halfeye

Half-eye (density + interval) plot (shortcut stat)

Description

Equivalent to `stat_slabinterval()`, whose default settings create half-eye (density + interval) plots.

Usage

stat_halfeye(
  mapping = NULL,
  data = NULL,
  geom = "slabinterval",
  position = "identity",
  ...
  slab_type = "pdf",
  p_limits = c(NA, NA),
  adjust = 1,
  trim = TRUE,
expand = FALSE,
breaks = "Sturges",
outline_bars = FALSE,
point_interval = "median_qi",
limits = NULL,
n = 501,
.width = c(0.66, 0.95),
orientation = NA,
na.rm = FALSE,
show.legend = c(size = FALSE),
inherit.aes = TRUE
)

Arguments

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

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

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

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

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

geom Use to override the default connection between stat_halfeye() and geom_slabinterval()

position Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" (position_dodge()) or "dodgejust" (position_dodgejust()) can be useful if you have overlapping geometries.

... Other arguments passed to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3 (see Aesthetics, below). They may also be parameters to the paired geom/stat. When paired with the default geom, geom_slabinterval(), these include:

normalize How to normalize heights of functions input to the thickness aesthetic. One of:

- "all": normalize so that the maximum height across all data is 1.
- "panels": normalize within panels so that the maximum height in each panel is 1.
- "xy": normalize within the x/y axis opposite the orientation of this geom so that the maximum height at each value of the opposite axis is 1.
- "groups": normalize within values of the opposite axis and within each group so that the maximum height in each group is 1.
• "none": values are taken as is with no normalization (this should probably only be used with functions whose values are in [0,1], such as CDFs).

fill_type What type of fill to use when the fill color or alpha varies within a slab. One of:
• "segments": breaks up the slab geometry into segments for each unique combination of fill color and alpha value. This approach is supported by all graphics devices and works well for sharp cutoff values, but can give ugly results if a large number of unique fill colors are being used (as in gradients, like in `stat_gradientinterval()`).
• "gradient": a `grid::linearGradient()` is used to create a smooth gradient fill. This works well for large numbers of unique fill colors, but requires R >= 4.1 and is not yet supported on all graphics devices. As of this writing, the `png()` graphics device with type = "cairo", the `svg()` device, the `pdf()` device, and the `ragg::agg_png()` devices are known to support this option. On R < 4.1, this option will fall back to `fill_type = "segment"` with a message.
• "auto": attempts to use `fill_type = "gradient"` if support for it can be auto-detected. On R >= 4.2, support for gradients can be auto-detected on some graphics devices; if support is not detected, this option will fall back to `fill_type = "segments"` (in case of a false negative, `fill_type = "gradient"` can be set explicitly). On R < 4.2, support for gradients cannot be auto-detected, so this will always fall back to `fill_type = "segments"`, in which case you can set `fill_type = "gradient"` explicitly if you are using a graphics device that supports gradients.

interval_size_domain A length-2 numeric vector giving the minimum and maximum of the values of the size aesthetic that will be translated into actual sizes for intervals drawn according to `interval_size_range` (see the documentation for that argument.)

interval_size_range A length-2 numeric vector. This geom scales the raw size aesthetic values when drawing interval and point sizes, as they tend to be too thick when using the default settings of `scale_size_continuous()`, which give sizes with a range of c(1, 6). The `interval_size_domain` value indicates the input domain of raw size values (typically this should be equal to the value of the range argument of the `scale_size_continuous()` function), and `interval_size_range` indicates the desired output range of the size values (the min and max of the actual sizes used to draw intervals). Most of the time it is not recommended to change the value of this argument, as it may result in strange scaling of legends; this argument is a holdover from earlier versions that did not have size aesthetics targeting the point and interval separately. If you want to adjust the size of the interval or points separately, you can instead use the `interval_size` or `point_size` aesthetics; see scales.

fatten_point A multiplicative factor used to adjust the size of the point relative to the size of the thickest interval line. If you wish to specify point sizes directly, you can also use the `point_size` aesthetic and `scale_point_size_continuous()`
or scale_point_size_discrete(); sizes specified with that aesthetic will not be adjusted using fatten_point.

**slab_type**

The type of slab function to calculate: probability density (or mass) function ("pdf"), cumulative distribution function ("cdf"), or complementary CDF ("ccdf").

**p_limits**

Probability limits (as a vector of size 2) used to determine the lower and upper limits of the slab. E.g., if this is \(c(0.001, 0.999)\), then a slab is drawn for the distribution from the quantile at \(p = 0.001\) to the quantile at \(p = 0.999\). If the lower (respectively upper) limit is NA, then the lower (upper) limit will be the minimum (maximum) of the distribution's support if it is finite, and 0.001 (0.999) if it is not finite. E.g., if p_limits is \(c(NA, NA)\) on a gamma distribution the effective value of p_limits would be \(c(0, 0.999)\) since the gamma distribution is defined on \((0, \infty)\); whereas on a normal distribution it would be equivalent to \(c(0.001, 0.999)\) since the normal distribution is defined on \((-\infty, \infty)\).

**adjust**

If slab_type is "pdf", bandwidth for the density estimator for sample data is adjusted by multiplying it by this value. See density() for more information.

**trim**

For sample data, should the density estimate be trimmed to the range of the input data? Default TRUE.

**expand**

For sample data, should the slab be expanded to the limits of the scale? Default FALSE. Can be length two to control expansion to the lower and upper limit respectively.

**breaks**

If slab_type is "histogram", the breaks parameter that is passed to hist() to determine where to put breaks in the histogram (for sample data).

**outline_bars**

For sample data (if slab_type is "histogram") and for discrete analytical distributions (whose slabs are drawn as histograms), determines if outlines in between the bars are drawn when the slab_color aesthetic is used. If FALSE (the default), the outline is drawn only along the tops of the bars; if TRUE, outlines in between bars are also drawn.

**point_interval**

A function from the point_interval() family (e.g., median_qi, mean_qi, mode_hdi, etc), or a string giving the name of a function from that family (e.g., "median_qi", "mean_qi", "mode_hdi", etc; if a string, the caller's environment is searched for the function, followed by the ggdist environment). This function determines the point summary (typically mean, median, or mode) and interval type (quantile interval, qi; highest-density interval, hdi; or highest-density continuous interval, hdci). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of orientation. See the point_interval() family of functions for more information.

**limits**

Manually-specified limits for the slab, as a vector of length two. These limits are combined with those computed based on p_limits as well as the limits defined by the scales of the plot to determine the limits used to draw the slab functions: these limits specify the maximal limits; i.e., if specified, the limits will not be wider than these (but may be narrower). Use NA to leave a limit alone; e.g. limits = \(c(\emptyset, NA)\) will ensure that the lower limit does not go below 0, but let the upper limit be determined by either p_limits or the scale settings.

**n**

Number of points at which to evaluate the function that defines the slab.

**.width**

The .width argument passed to point_interval: a vector of probabilities to use that determine the widths of the resulting intervals. If multiple probabilities
are provided, multiple intervals per group are generated, each with a different probability interval (and value of the corresponding .width and level generated variables).

**orientation**

Whether this geom is drawn horizontally or vertically. One of:

- NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
- "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify different groups. For each group, uses the x, xmin, xmax, and thickness aesthetics to draw points, intervals, and slabs.
- "vertical" (or "x"): draw vertically, using the x aesthetic to identify different groups. For each group, uses the y, ymin, ymax, and thickness aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for orientation, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal" (tidybayes had an orientation parameter before base ggplot did, hence the discrepancy).

**na.rm**

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

**show.legend**

Should this layer be included in the legends? Default is c(size = FALSE), unlike most geoms, to match its common use cases. FALSE hides all legends, TRUE shows all legends, and NA shows only those that are mapped (the default for most geoms).

**inherit.aes**

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

**Details**

To visualize sample data, such as a data distribution, samples from a bootstrap distribution, or a Bayesian posterior, you can supply samples to the x or y aesthetic.

To visualize analytical distributions, you can use the xdist or ydist aesthetic. For historical reasons, you can also use dist to specify the distribution, though this is not recommended as it does not work as well with orientation detection. These aesthetics can be used as follows:

- xdist, ydist, and dist can be any distribution object from the distributional package (dist_normal(), dist_beta(), etc) or can be a posterior::rvar() object. Since these functions are vectorized, other columns can be passed directly to them in an aes() specification; e.g. `aes(dist = dist_normal(mu, sigma))` will work if mu and sigma are columns in the input data frame.
- dist can be a character vector giving the distribution name. Then the arg1, ... arg9 aesthetics (or args as a list column) specify distribution arguments. Distribution names should correspond to R functions that have "p", "q", and "d" functions; e.g. "norm" is a valid distribution name because R defines the pnorm(), qnorm(), and dnorm() functions for Normal distributions.

See the parse_dist() function for a useful way to generate dist and args values from human-readable distribution specs (like "normal(0,1)"). Such specs are also produced by other packages (like the brms:::get_prior function in brms); thus, parse_dist() combined with the stats described here can help you visualize the output of those functions.
Value

A `ggplot2::Stat` representing a half-eye (density + interval) geometry which can be added to a `ggplot()` object.

Computed Variables

The following variables are computed by this stat and made available for use in aesthetic specifications (`aes()`) using the `stat()` or `after_stat()` functions:

- `x` or `y`: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is `x` or `y` depends on orientation
- `xmin` or `ymin`: For intervals, the lower end of the interval from the interval function.
- `xmax` or `ymax`: For intervals, the upper end of the interval from the interval function.
- `.width`: For intervals, the interval width as a numeric value in `[0, 1]`. For slabs, the width of the smallest interval containing that value of the slab.
- `level`: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.
- `pdf`: For slabs, the probability density function (PDF). If `options("ggdist.experimental.slab_data_in_intervals")` is `TRUE`: For intervals, the PDF at the point summary; intervals also have `pdf_min` and `pdf_max` for the PDF at the lower and upper ends of the interval.
- `cdf`: For slabs, the cumulative distribution function. If `options("ggdist.experimental.slab_data_in_intervals")` is `TRUE`: For intervals, the CDF at the point summary; intervals also have `cdf_min` and `cdf_max` for the CDF at the lower and upper ends of the interval.
- `f`: For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by `slab_type`.
- `n`: For slabs, the number of data points summarized into that slab. If the slab was created from an analytical distribution via the `xdist`, `ydist`, or `dist` aesthetic, `n` will be `Inf`.

Aesthetics

The slab+interval stats and geoms have a wide variety of aesthetics that control the appearance of their three sub-geometries: the slab, the point, and the interval.

These stats support the following aesthetics:

- `x`: x position of the geometry (when orientation = "vertical"); or sample data to be summarized (when orientation = "horizontal" with sample data).
- `y`: y position of the geometry (when orientation = "horizontal"); or sample data to be summarized (when orientation = "vertical" with sample data).
- `xdist`: When using analytical distributions, distribution to map on the x axis: a `distributional` object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `ydist`: When using analytical distributions, distribution to map on the y axis: a `distributional` object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `dist`: When using analytical distributions, a name of a distribution (e.g. "norm"), a `distributional` object (e.g. `dist_normal()`) or a `posterior::rvar()` object. See Details.
- `args`: Distribution arguments (args or arg1, ... arg9). See Details.
In addition, in their default configuration (paired with `geom_slabinterval()`) the following aesthetics are supported by the underlying geom:

**Slab-specific aesthetics**

- **thickness**: The thickness of the slab at each x value (if orientation = "horizontal") or y value (if orientation = "vertical") of the slab.
- **side**: Which side to place the slab on. "topright", "top", and "right" are synonyms which cause the slab to be drawn on the top or the right depending on if orientation is "horizontal" or "vertical". "bottomleft", "bottom", and "left" are synonyms which cause the slab to be drawn on the bottom or the left depending on if orientation is "horizontal" or "vertical". "topleft" causes the slab to be drawn on the top or the left, and "bottomright" causes the slab to be drawn on the bottom or the right. "both" draws the slab mirrored on both sides (as in a violin plot).
- **scale**: What proportion of the region allocated to this geom to use to draw the slab. If scale = 1, slabs that use the maximum range will just touch each other. Default is 0.9 to leave some space.
- **justification**: Justification of the interval relative to the slab, where 0 indicates bottom/left justification and 1 indicates top/right justification (depending on orientation). If justification is NULL (the default), then it is set automatically based on the value of side: when side is "top"/"right" justification is set to 0, when side is "bottom"/"left" justification is set to 1, and when side is "both" justification is set to 0.5.
- **datatype**: When using composite geoms directly without a stat (e.g. `geom_slabinterval()`), datatype is used to indicate which part of the geom a row in the data targets: rows with datatype = "slab" target the slab portion of the geometry and rows with datatype = "interval" target the interval portion of the geometry. This is set automatically when using `ggdist` stats.

**Interval-specific aesthetics**

- **xmin**: Left end of the interval sub-geometry (if orientation = "horizontal").
- **xmax**: Right end of the interval sub-geometry (if orientation = "horizontal").
- **ymin**: Lower end of the interval sub-geometry (if orientation = "vertical").
- **ymax**: Upper end of the interval sub-geometry (if orientation = "vertical").

**Point-specific aesthetics**

- **shape**: Shape type used to draw the point sub-geometry.

**Color aesthetics**

- **colour**: (or color) The color of the interval and point sub-geometries. Use the slab_color, interval_color, or point_color aesthetics (below) to set sub-geometry colors separately.
- **fill**: The fill color of the slab and point sub-geometries. Use the slab_fill or point_fill aesthetics (below) to set sub-geometry colors separately.
- **alpha**: The opacity of the slab, interval, and point sub-geometries. Use the slab_alpha, interval_alpha, or point_alpha aesthetics (below) to set sub-geometry colors separately.
- **colour_ramp**: (or color_ramp) A secondary scale that modifies the color scale to "ramp" to another color. See `scale_colour_ramp()` for examples.
• fill_ramp: A secondary scale that modifies the fill scale to "ramp" to another color. See `scale_fill_ramp()` for examples.

Line aesthetics

• size: Width of the outline around the slab (if visible). Also determines the width of the line used to draw the interval and the size of the point, but raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the slab_size, interval_size, or point_size aesthetics (below) to set sub-geometry line widths separately (note that when size is set directly using the override aesthetics, interval and point sizes are not affected by interval_size_domain, interval_size_range, and fatten_point).

• stroke: Width of the outline around the point sub-geometry.

• linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the interval and the outline of the slab (if it is visible). Use the slab_linetype or interval_linetype aesthetics (below) to set sub-geometry line types separately.

Slab-specific color/line override aesthetics

• slab_fill: Override for fill: the fill color of the slab.

• slab_colour: (or slab_color) Override for colour/color: the outline color of the slab.

• slab_alpha: Override for alpha: the opacity of the slab.

• slab_size: Override for size: the width of the outline of the slab.

• slab_linetype: Override for linetype: the line type of the outline of the slab.

Interval-specific color/line override aesthetics

• interval_colour: (or interval_color) Override for colour/color: the color of the interval.

• interval_alpha: Override for alpha: the opacity of the interval.

• interval_size: Override for size: the line width of the interval.

• interval_linetype: Override for linetype: the line type of the interval.

Point-specific color/line override aesthetics

• point_fill: Override for fill: the fill color of the point.

• point_colour: (or point_color) Override for colour/color: the outline color of the point.

• point_alpha: Override for alpha: the opacity of the point.

• point_size: Override for size: the size of the point.

Other aesthetics (these work as in standard geoms)

• width

• height

• group

See examples of some of these aesthetics in action in vignette("slabinterval"). Learn more about the sub-geom override aesthetics (like interval_color) in the scales documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").
See Also

See `geom_slabinterval()` for the geom underlying this stat. See `stat_slabinterval()` for the stat this shortcut is based on.

Other slabinterval stats: `stat_ccdfinterval()`, `stat_cdfinterval()`, `stat_eye()`, `stat_gradientinterval()`, `stat_histinterval()`, `stat_interval()`, `stat_pointinterval()`, `stat_slab()`

Examples

```r
library(dplyr)
library(ggplot2)
library(distributional)

theme_set(theme_ggdist())

# ON SAMPLE DATA
set.seed(1234)
df = data.frame(
  group = c("a", "b", "c"),
  value = rnorm(1500, mean = c(5, 7, 9), sd = c(1, 1.5, 1))
)
df %>%
ggplot(aes(x = value, y = group)) +
stat_halfeye()

# ON ANALYTICAL DISTRIBUTIONS
dist_df = data.frame(
  group = c("a", "b", "c"),
  mean = c(5, 7, 8),
  sd = c(1, 1.5, 1)
)
# Vectorized distribution types, like distributional::dist_normal()
# and posterior::rvar(), can be used with the `xdist` / `ydist` aesthetics
dist_df %>%
ggplot(aes(y = group, xdist = dist_normal(mean, sd))) +
stat_halfeye()
```

---

**stat_histinterval**  
Histogram + interval plot (shortcut stat)

**Description**

Shortcut version of `stat_slabinterval()` with `geom_slabinterval()` for creating histogram + interval plots.

Roughly equivalent to:

```r
stat_slabinterval(
  slab_type = "histogram"
)
```
Usage

```r
stat_histinterval(
  mapping = NULL,
  data = NULL,
  geom = "slabinterval",
  position = "identity",
  ..., 
  slab_type = "histogram",
  p_limits = c(NA, NA),
  adjust = 1,
  trim = TRUE,
  expand = FALSE,
  breaks = "Sturges",
  outline_bars = FALSE,
  point_interval = "median_qi",
  limits = NULL,
  n = 501,
  .width = c(0.66, 0.95),
  orientation = NA,
  na.rm = FALSE,
  show.legend = c(size = FALSE),
  inherit.aes = TRUE
)
```

Arguments

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

- **data**
  The data to be displayed in this layer. There are three options:
  - If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
  - A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a `data.frame`. See `fortify()` for which variables will be created.
  - A function will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x, 10)`).

- **geom**
  Use to override the default connection between `stat_histinterval()` and `geom_slabinterval()`.

- **position**
  Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" (`position_dodge()`) or "dodgejust" (`position_dodgejust()`) can be useful if you have overlapping geometries.

- **...**
  Other arguments passed to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3` (see `Aesthetics`, below). They may also be parameters to the paired geom/stat. When paired with the default geom, `geom_slabinterval()`, these include:
normalize  How to normalize heights of functions input to the thickness aesthetic. One of:

- "all": normalize so that the maximum height across all data is 1.
- "panels": normalize within panels so that the maximum height in each panel is 1.
- "xy": normalize within the x/y axis opposite the orientation of this geom so that the maximum height at each value of the opposite axis is 1.
- "groups": normalize within values of the opposite axis and within each group so that the maximum height in each group is 1.
- "none": values are taken as is with no normalization (this should probably only be used with functions whose values are in [0,1], such as CDFs).

fill_type  What type of fill to use when the fill color or alpha varies within a slab. One of:

- "segments": breaks up the slab geometry into segments for each unique combination of fill color and alpha value. This approach is supported by all graphics devices and works well for sharp cutoff values, but can give ugly results if a large number of unique fill colors are being used (as in gradients, like in `stat_gradientinterval()`).
- "gradient": a `grid::linearGradient()` is used to create a smooth gradient fill. This works well for large numbers of unique fill colors, but requires R >= 4.1 and is not yet supported on all graphics devices. As of this writing, the `png()` graphics device with type = "cairo", the `svg()` device, the `pdf()` device, and the `ragg::agg_png()` devices are known to support this option. On R < 4.1, this option will fall back to `fill_type = "segment"` with a message.
- "auto": attempts to use `fill_type = "gradient"` if support for it can be auto-detected. On R >= 4.2, support for gradients can be auto-detected on some graphics devices; if support is not detected, this option will fall back to `fill_type = "segments"` (in case of a false negative, `fill_type = "gradient"` can be set explicitly). On R < 4.2, support for gradients cannot be auto-detected, so this will always fall back to `fill_type = "segments"`, in which case you can set `fill_type = "gradient"` explicitly if you are using a graphics device that support gradients.

interval_size_domain  A length-2 numeric vector giving the minimum and maximum of the values of the size aesthetic that will be translated into actual sizes for intervals drawn according to `interval_size_range` (see the documentation for that argument.)

interval_size_range  A length-2 numeric vector. This geom scales the raw size aesthetic values when drawing interval and point sizes, as they tend to be too thick when using the default settings of `scale_size_continuous()`, which give sizes with a range of c(1, 6). The `interval_size_domain` value indicates the input domain of raw size values (typically this should be equal to the value of the range argument of the `scale_size_continuous()` function), and `interval_size_range` indicates the desired output range of
the size values (the min and max of the actual sizes used to draw intervals). Most of the time it is not recommended to change the value of this argument, as it may result in strange scaling of legends; this argument is a holdover from earlier versions that did not have size aesthetics targeting the point and interval separately. If you want to adjust the size of the interval or points separately, you can instead use the interval_size or point_size aesthetics; see scales.

fatten_point A multiplicative factor used to adjust the size of the point relative to the size of the thickest interval line. If you wish to specify point sizes directly, you can also use the point_size aesthetic and scale_point_size_continuous() or scale_point_size_discrete(); sizes specified with that aesthetic will not be adjusted using fatten_point.

slab_type The type of slab function to calculate: probability density (or mass) function ("pdf"), cumulative distribution function ("cdf"), or complementary CDF ("ccdf").

p_limits Probability limits (as a vector of size 2) used to determine the lower and upper limits of the slab. E.g., if this is c(.001, .999), then a slab is drawn for the distribution from the quantile at p = .001 to the quantile at p = .999. If the lower (respectively upper) limit is NA, then the lower (upper) limit will be the minimum (maximum) of the distribution’s support if it is finite, and 0 .001 (.999) if it is not finite. E.g., if p_limits is c(NA, NA) on a gamma distribution the effective value of p_limits would be c(0, .999) since the gamma distribution is defined on (0, Inf); whereas on a normal distribution it would be equivalent to c(.001, .999) since the normal distribution is defined on (-Inf, Inf).

adjust If slab_type is "pdf", bandwidth for the density estimator for sample data is adjusted by multiplying it by this value. See density() for more information.

trim For sample data, should the density estimate be trimmed to the range of the input data? Default TRUE.

expand For sample data, should the slab be expanded to the limits of the scale? Default FALSE. Can be length two to control expansion to the lower and upper limit respectively.

breaks If slab_type is "histogram", the breaks parameter that is passed to hist() to determine where to put breaks in the histogram (for sample data).

outline_bars For sample data (if slab_type is "histogram") and for discrete analytical distributions (whose slabs are drawn as histograms), determines if outlines in between the bars are drawn when the slab_color aesthetic is used. If FALSE (the default), the outline is drawn only along the tops of the bars; if TRUE, outlines in between bars are also drawn.

point_interval A function from the point_interval() family (e.g., median_qi, mean_qi, mode_hdi, etc), or a string giving the name of a function from that family (e.g., "median_qi", "mean_qi", "mode_hdi", etc; if a string, the caller's environment is searched for the function, followed by the ggdist environment). This function determines the point summary (typically mean, median, or mode) and interval type (quantile interval, qi; highest-density interval, hdi; or highest-density continuous interval, hdci). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of orientation. See the point_interval() family of functions for more information.
Manually-specified limits for the slab, as a vector of length two. These limits are combined with those computed based on `p_limits` as well as the limits defined by the scales of the plot to determine the limits used to draw the slab functions: these limits specify the maximal limits; i.e., if specified, the limits will not be wider than these (but may be narrower). Use NA to leave a limit alone; e.g. `limits = c(0, NA)` will ensure that the lower limit does not go below 0, but let the upper limit be determined by either `p_limits` or the scale settings.

Number of points at which to evaluate the function that defines the slab.

The `.width` argument passed to `point_interval`: a vector of probabilities to use that determine the widths of the resulting intervals. If multiple probabilities are provided, multiple intervals per group are generated, each with a different probability interval (and value of the corresponding `.width` and `level` generated variables).

Whether this geom is drawn horizontally or vertically. One of:

- `NA` (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
- "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify different groups. For each group, uses the x, xmin, xmax, and thickness aesthetics to draw points, intervals, and slabs.
- "vertical" (or "x"): draw vertically, using the x aesthetic to identify different groups. For each group, uses the y, ymin, ymax, and thickness aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for `orientation`, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal" (tidybayes had an orientation parameter before base ggplot did, hence the discrepancy).

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

Should this layer be included in the legends? Default is `c(size = FALSE)`, unlike most geoms, to match its common use cases. FALSE hides all legends, TRUE shows all legends, and NA shows only those that are mapped (the default for most geoms).

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

**Details**

To visualize sample data, such as a data distribution, samples from a bootstrap distribution, or a Bayesian posterior, you can supply samples to the x or y aesthetic.

To visualize analytical distributions, you can use the xdist or ydist aesthetic. For historical reasons, you can also use dist to specify the distribution, though this is not recommended as it does not work as well with orientation detection. These aesthetics can be used as follows:

- xdist, ydist, and dist can be any distribution object from the `distributional` package (`dist_normal()`, `dist_beta()`, etc) or can be a `posterior::rvar()` object. Since these functions are vector-
ized, other columns can be passed directly to them in an `aes()` specification; e.g. `aes(dist = dist_normal(mu, sigma))` will work if mu and sigma are columns in the input data frame.

- `dist` can be a character vector giving the distribution name. Then the `arg1, ... arg9` aesthetics (or `args` as a list column) specify distribution arguments. Distribution names should correspond to R functions that have "p", "q", and "d" functions; e.g. "norm" is a valid distribution name because R defines the `pnorm()`, `qnorm()`, and `dnorm()` functions for Normal distributions.

See the `parse_dist()` function for a useful way to generate `dist` and `args` values from human-readable distribution specs (like "normal(0,1)"). Such specs are also produced by other packages (like the `brms::get_prior` function in brms); thus, `parse_dist()` combined with the stats described here can help you visualize the output of those functions.

**Value**

A `ggplot2::Stat` representing a histogram + interval geometry which can be added to a `ggplot()` object.

**Computed Variables**

The following variables are computed by this stat and made available for use in aesthetic specifications (`aes()`) using the `stat()` or `after_stat()` functions:

- `x` or `y`: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is `x` or `y` depends on orientation.
- `xmin` or `ymin`: For intervals, the lower end of the interval from the interval function.
- `xmax` or `ymax`: For intervals, the upper end of the interval from the interval function.
- `.width`: For intervals, the interval width as a numeric value in `[0, 1]`. For slabs, the width of the smallest interval containing that value of the slab.
- `level`: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.
- `pdf`: For slabs, the probability density function (PDF). If `options("ggdist.experimental.slab_data_in_intervals")` is `TRUE`: For intervals, the PDF at the point summary; intervals also have `pdf_min` and `pdf_max` for the PDF at the lower and upper ends of the interval.
- `cdf`: For slabs, the cumulative distribution function. If `options("ggdist.experimental.slab_data_in_intervals")` is `TRUE`: For intervals, the CDF at the point summary; intervals also have `cdf_min` and `cdf_max` for the CDF at the lower and upper ends of the interval.
- `f`: For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by `slab_type`.
- `n`: For slabs, the number of data points summarized into that slab. If the slab was created from an analytical distribution via the `xdist`, `ydist`, or `dist` aesthetic, `n` will be `Inf`.

**Aesthetics**

The slab+interval stats and geoms have a wide variety of aesthetics that control the appearance of their three sub-geometries: the `slab`, the `point`, and the `interval`.

These stats support the following aesthetics:
• **x**: x position of the geometry (when orientation = "vertical"); or sample data to be summarized (when orientation = "horizontal" with sample data).

• **y**: y position of the geometry (when orientation = "horizontal"); or sample data to be summarized (when orientation = "vertical" with sample data).

• **xdist**: When using analytical distributions, distribution to map on the x axis: a `distributional` object (e.g. `dist_normal()`) or a `posterior::rvar()` object.

• **ydist**: When using analytical distributions, distribution to map on the y axis: a `distributional` object (e.g. `dist_normal()`) or a `posterior::rvar()` object.

• **dist**: When using analytical distributions, a name of a distribution (e.g. "norm"), a `distributional` object (e.g. `dist_normal()`), or a `posterior::rvar()` object. See Details.

• **args**: Distribution arguments (args or arg1, ... arg9). See Details.

In addition, in their default configuration (paired with `geom_slabinterval()`) the following aesthetics are supported by the underlying geom:

**Slab-specific aesthetics**

• **thickness**: The thickness of the slab at each x value (if orientation = "horizontal") or y value (if orientation = "vertical") of the slab.

• **side**: Which side to place the slab on. "topright", "top", and "right" are synonyms which cause the slab to be drawn on the top or the right depending on if orientation is "horizontal" or "vertical". "bottomleft", "bottom", and "left" are synonyms which cause the slab to be drawn on the bottom or the left depending on if orientation is "horizontal" or "vertical". "topleft" causes the slab to be drawn on the top or the left, and "bottomright" causes the slab to be drawn on the bottom or the right. "both" draws the slab mirrored on both sides (as in a violin plot).

• **scale**: What proportion of the region allocated to this geom to use to draw the slab. If scale = 1, slabs that use the maximum range will just touch each other. Default is 0.9 to leave some space.

• **justification**: Justification of the interval relative to the slab, where 0 indicates bottom/left justification and 1 indicates top/right justification (depending on orientation). If justification is NULL (the default), then it is set automatically based on the value of side: when side is "top"/"right" justification is set to 0, when side is "bottom"/"left" justification is set to 1, and when side is "both" justification is set to 0.5.

• **datatype**: When using composite geoms directly without a `stat` (e.g. `geom_slabinterval()`), datatype is used to indicate which part of the geom a row in the data targets: rows with datatype = "slab" target the slab portion of the geometry and rows with datatype = "interval" target the interval portion of the geometry. This is set automatically when using ggdist stats.

**Interval-specific aesthetics**

• **xmin**: Left end of the interval sub-geometry (if orientation = "horizontal").

• **xmax**: Right end of the interval sub-geometry (if orientation = "horizontal").

• **ymin**: Lower end of the interval sub-geometry (if orientation = "vertical").

• **ymax**: Upper end of the interval sub-geometry (if orientation = "vertical").

**Point-specific aesthetics**
• shape: Shape type used to draw the point sub-geometry.

Color aesthetics

• colour: (or color) The color of the interval and point sub-geometries. Use the slab_color, interval_color, or point_color aesthetics (below) to set sub-geometry colors separately.
• fill: The fill color of the slab and point sub-geometries. Use the slab_fill or point_fill aesthetics (below) to set sub-geometry colors separately.
• alpha: The opacity of the slab, interval, and point sub-geometries. Use the slab_alpha, interval_alpha, or point_alpha aesthetics (below) to set sub-geometry colors separately.
• colour_ramp: (or color_ramp) A secondary scale that modifies the color scale to "ramp" to another color. See scale_colour_ramp() for examples.
• fill_ramp: A secondary scale that modifies the fill scale to "ramp" to another color. See scale_fill_ramp() for examples.

Line aesthetics

• size: Width of the outline around the slab (if visible). Also determines the width of the line used to draw the interval and the size of the point, but raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the slab_size, interval_size, or point_size aesthetics (below) to set sub-geometry line widths separately (note that when size is set directly using the override aesthetics, interval and point sizes are not affected by interval_size_domain, interval_size_range, and fatten_point).
• stroke: Width of the outline around the point sub-geometry.
• linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the interval and the outline of the slab (if it is visible). Use the slab_linetype or interval_linetype aesthetics (below) to set sub-geometry line types separately.

Slab-specific color/line override aesthetics

• slab_fill: Override for fill: the fill color of the slab.
• slab_colour: (or slab_color) Override for colour/color: the outline color of the slab.
• slab_alpha: Override for alpha: the opacity of the slab.
• slab_size: Override for size: the width of the outline of the slab.
• slab_linetype: Override for linetype: the line type of the outline of the slab.

Interval-specific color/line override aesthetics

• interval_colour: (or interval_color) Override for colour/color: the color of the interval.
• interval_alpha: Override for alpha: the opacity of the interval.
• interval_size: Override for size: the line width of the interval.
• interval_linetype: Override for linetype: the line type of the interval.

Point-specific color/line override aesthetics
stat_histinterval

- **point_fill**: Override for fill: the fill color of the point.
- **point_colour**: (or point_color) Override for colour/color: the outline color of the point.
- **point_alpha**: Override for alpha: the opacity of the point.
- **point_size**: Override for size: the size of the point.

**Other aesthetics** (these work as in standard geoms)

- width
- height
- group

See examples of some of these aesthetics in action in vignette("slabinterval"). Learn more about the sub-geom override aesthetics (like interval_color) in the scales documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").

**See Also**

See **geom_slabinterval()** for the geom underlying this stat. See **stat_slabinterval()** for the stat this shortcut is based on.

Other slabinterval stats: **stat_ccdfinterval()**, **stat_cdfinterval()**, **stat_eye()**, **stat_gradientinterval()**, **stat_halfeye()**, **stat_interval()**, **stat_pointinterval()**, **stat_slab()**

**Examples**

```r
library(dplyr)
library(ggplot2)
library(distributional)

theme_set(theme_ggdist())

# ON SAMPLE DATA
set.seed(1234)
df = data.frame(
  group = c("a", "b", "c"),
  value = rnorm(1500, mean = c(5, 7, 9), sd = c(1, 1.5, 1))
)
df %>%
ggplot(aes(x = value, y = group)) +
stat_histinterval()

# ON ANALYTICAL DISTRIBUTIONS
dist_df = data.frame(
  group = c("a", "b", "c"),
  mean = c(5, 7, 8),
  sd = c(1, 1.5, 1)
)
# Vectorized distribution types, like distributional::dist_normal()
# and posterior::rvar(), can be used with the `xdist` / `ydist` aesthetics
dist_df %>%
ggplot(aes(y = group, xdist = dist_normal(mean, sd))) +
stat_histinterval()
```
Description

Shortcut version of `stat_slabinterval()` with `geom_interval()` for creating multiple-interval plots.

Roughly equivalent to:

```r
stat_slabinterval(
  aes(colour = stat(level), size = NULL),
  geom = "interval",
  .width = c(0.5, 0.8, 0.95), show_slab = FALSE,
  show.legend = NA
)
```

Usage

```r
stat_interval(
  mapping = NULL,
  data = NULL,
  geom = "interval",
  position = "identity",
  ..., 
  .width = c(0.5, 0.8, 0.95),
  point_interval = "median_qi",
  orientation = NA,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

- `mapping` Set of aesthetic mappings created by `aes()` or `aes_()`. If specified and `inherit.aes` is `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_interval

geom

Use to override the default connection between `stat_interval()` and `geom_interval()`

position

Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" (`position_dodge()`) or "dodgejust" (`position_dodgejust()`) can be useful if you have overlapping geometries.

... Other arguments passed to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3` (see `Aesthetics`, below). They may also be parameters to the paired geom/stat. When paired with the default geom, `geom_interval()`, these include:

interval_size_range A length-2 numeric vector. This geom scales the raw size aesthetics when drawing interval and point sizes, as they tend to be too thick when using the default settings of `scale_size_continuous()`. which give sizes with a range of c(1, 6). The interval_size_domain value indicates the input domain of raw size values (typically this should be equal to the value of the range argument of the `scale_size_continuous()` function), and interval_size_range indicates the desired output range of the size values (the min and max of the actual sizes used to draw intervals). Most of the time it is not recommended to change the value of this argument, as it may result in strange scaling of legends; this argument is a holdover from earlier versions that did not have size aesthetics targeting the point and interval separately. If you want to adjust the size of the interval or points separately, you can instead use the interval_size or point_size aesthetics; see `scales`.

interval_size_domain A length-2 numeric vector giving the minimum and maximum of the values of the size aesthetic that will be translated into actual sizes for intervals drawn according to `interval_size_range` (see the documentation for that argument.)

.width

The .width argument passed to `point_interval`: a vector of probabilities to use that determine the widths of the resulting intervals. If multiple probabilities are provided, multiple intervals per group are generated, each with a different probability interval (and value of the corresponding .width and level generated variables).

point_interval A function from the `point_interval()` family (e.g., `median_qi`, `mean_qi`, `mode_hdi`, etc), or a string giving the name of a function from that family (e.g., "median_qi", "mean_qi", "mode_hdi", etc; if a string, the caller's environment is searched for the function, followed by the `ggdist` environment). This function determines the point summary (typically mean, median, or mode) and interval type (quantile interval, qi; highest-density interval, hdi; or highest-density continuous interval, hdci). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of orientation. See the `point_interval()` family of functions for more information.

orientation Whether this geom is drawn horizontally or vertically. One of:

- NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
- "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify different groups. For each group, uses the x, xmin, xmax, and thickness aesthetics to draw points, intervals, and slabs.
• "vertical" (or "x"): draw vertically, using the x aesthetic to identify different groups. For each group, uses the y, ymin, ymax, and thickness aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for orientation, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal" (tidybayes had an orientation parameter before base ggplot did, hence the discrepancy).

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

show.legend
Should this layer be included in the legends? Default is c(size = FALSE), unlike most geoms, to match its common use cases. FALSE hides all legends, TRUE shows all legends, and NA shows only those that are mapped (the default for most geoms).

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

Details

To visualize sample data, such as a data distribution, samples from a bootstrap distribution, or a Bayesian posterior, you can supply samples to the x or y aesthetic.

To visualize analytical distributions, you can use the xdist or ydist aesthetic. For historical reasons, you can also use dist to specify the distribution, though this is not recommended as it does not work as well with orientation detection. These aesthetics can be used as follows:

• xdist, ydist, and dist can be any distribution object from the distributional package (dist_normal(), dist_beta(), etc) or can be a posterior::rvar() object. Since these functions are vectorized, other columns can be passed directly to them in an aes() specification; e.g. aes(dist = dist_normal(mu, sigma)) will work if mu and sigma are columns in the input data frame.

• dist can be a character vector giving the distribution name. Then the arg1, ... arg9 aesthetics (or args as a list column) specify distribution arguments. Distribution names should correspond to R functions that have "p", "q", and "d" functions; e.g. "norm" is a valid distribution name because R defines the pnorm(), qnorm(), and dnorm() functions for Normal distributions. See the parse_dist() function for a useful way to generate dist and args values from human-readable distribution specs (like "normal(0,1)"). Such specs are also produced by other packages (like the brms::get_prior function in brms); thus, parse_dist() combined with the stats described here can help you visualize the output of those functions.

Value

A ggplot2::Stat representing a multiple-interval geometry which can be added to a ggplot() object.

Computed Variables

The following variables are computed by this stat and made available for use in aesthetic specifications (aes()) using the stat() or after_stat() functions:
• x or y: For slabs, the input values to the slab function. For intervals, the point summary from
the interval function. Whether it is x or y depends on orientation
• xmin or ymin: For intervals, the lower end of the interval from the interval function.
• xmax or ymax: For intervals, the upper end of the interval from the interval function.
• .width: For intervals, the interval width as a numeric value in [0, 1]. For slabs, the width
of the smallest interval containing that value of the slab.
• level: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest
interval containing that value of the slab.
• pdf: For slabs, the probability density function (PDF). If options("ggdist.experimental.slab_data_in_intervals"
is TRUE: For intervals, the PDF at the point summary; intervals also have pdf_min and pdf_max
for the PDF at the lower and upper ends of the interval.
• cdf: For slabs, the cumulative distribution function. If options("ggdist.experimental.slab_data_in_intervals"
is TRUE: For intervals, the CDF at the point summary; intervals also have cdf_min and cdf_max
for the CDF at the lower and upper ends of the interval.

Aesthetics

The slab+interval stats and geoms have a wide variety of aesthetics that control the appearance of
their three sub-geometries: the slab, the point, and the interval.

These stats support the following aesthetics:

• x: x position of the geometry (when orientation = "vertical"); or sample data to be summa-
rized (when orientation = "horizontal" with sample data).
• y: y position of the geometry (when orientation = "horizontal"); or sample data to be sum-
murized (when orientation = "vertical" with sample data).
• xdist: When using analytical distributions, distribution to map on the x axis: a distributional
object (e.g. dist_normal()) or a posterior::rvar() object.
• ydist: When using analytical distributions, distribution to map on the y axis: a distributional
object (e.g. dist_normal()) or a posterior::rvar() object.
• dist: When using analytical distributions, a name of a distribution (e.g. "norm"), a distributional
object (e.g. dist_normal()), or a posterior::rvar() object. See Details.
• args: Distribution arguments (args or arg1, ... arg9). See Details.

In addition, in their default configuration (paired with geom_interval()) the following aesthetics
are supported by the underlying geom:

Interval-specific aesthetics

• xmin: Left end of the interval sub-geometry (if orientation = "horizontal").
• xmax: Right end of the interval sub-geometry (if orientation = "horizontal").
• ymin: Lower end of the interval sub-geometry (if orientation = "vertical").
• ymax: Upper end of the interval sub-geometry (if orientation = "vertical").

Color aesthetics
• colour: (or color) The color of the interval and point sub-geometries. Use the slab_color, interval_color, or point_color aesthetics (below) to set sub-geometry colors separately.

• fill: The fill color of the slab and point sub-geometries. Use the slab_fill or point_fill aesthetics (below) to set sub-geometry colors separately.

• alpha: The opacity of the slab, interval, and point sub-geometries. Use the slab_alpha, interval_alpha, or point_alpha aesthetics (below) to set sub-geometry colors separately.

• colour_ramp: (or color_ramp) A secondary scale that modifies the color scale to "ramp" to another color. See scale_colour_ramp() for examples.

• fill_ramp: A secondary scale that modifies the fill scale to "ramp" to another color. See scale_fill_ramp() for examples.

Line aesthetics

• size: Width of the outline around the slab (if visible). Also determines the width of the line used to draw the interval and the size of the point, but raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the slab_size, interval_size, or point_size aesthetics (below) to set sub-geometry line widths separately (note that when size is set directly using the override aesthetics, interval and point sizes are not affected by interval_size_domain, interval_size_range, and fatten_point).

• stroke: Width of the outline around the point sub-geometry.

• linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the interval and the outline of the slab (if it is visible). Use the slab_linetype or interval_linetype aesthetics (below) to set sub-geometry line types separately.

Interval-specific color/line override aesthetics

• interval_colour: (or interval_color) Override for colour/color: the color of the interval.

• interval_alpha: Override for alpha: the opacity of the interval.

• interval_size: Override for size: the line width of the interval.

• interval_linetype: Override for linetype: the line type of the interval.

Other aesthetics (these work as in standard geoms)

• width

• height

• group

See examples of some of these aesthetics in action in vignette("slabinterval"). Learn more about the sub-geom override aesthetics (like interval_color) in the scales documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").

See Also

See geom_interval() for the geom underlying this stat. See stat_slabinterval() for the stat this shortcut is based on.

Other slabinterval stats: stat_ccdfinterval(), stat_cdfinterval(), stat_eye(), stat_gradientinterval(), stat_halfeye(), stat_histinterval(), stat_pointinterval(), stat_slab()
Examples

library(dplyr)
library(ggplot2)
library(distributional)

theme_set(theme_ggdist())

# ON SAMPLE DATA
set.seed(1234)
df = data.frame(
  group = c("a", "b", "c"),
  value = rnorm(1500, mean = c(5, 7, 9), sd = c(1, 1.5, 1))
)
df %>%
  ggplot(aes(x = value, y = group)) +
  stat_interval() +
  scale_color_brewer()

# ON ANALYTICAL DISTRIBUTIONS
dist_df = data.frame(
  group = c("a", "b", "c"),
  mean = c(5, 7, 8),
  sd = c(1, 1.5, 1)
)
# Vectorized distribution types, like distributional::dist_normal()
# and posterior::rvar(), can be used with the `xdist` / `ydist` aesthetics
dist_df %>%
  ggplot(aes(y = group, xdist = dist_normal(mean, sd))) +
  stat_interval() +
  scale_color_brewer

---

stat_lineribbon  
*Line + multiple-ribbon plot (shortcut stat)*

Description

A combination of `stat_slabinterval()` and `geom_lineribbon()` with sensible defaults for making line + multiple-ribbon plots. While `geom_lineribbon()` is intended for use on data frames that have already been summarized using a `point_interval()` function, `stat_lineribbon()` is intended for use directly on data frames of draws or of analytical distributions, and will perform the summarization using a `point_interval()` function.

Roughly equivalent to:

```r
stat_slabinterval(
  aes(group = stat(level), fill = stat(level), size = NULL),
  geom = "lineribbon",
  .width = c(0.5, 0.8, 0.95), show_slab = FALSE,
  show.legend = NA
)```

Usage

```r
stat_lineribbon(
  mapping = NULL,
  data = NULL,
  geom = "lineribbon",
  position = "identity",
  ...,
  .width = c(0.5, 0.8, 0.95),
  point_interval = "median_qi",
  orientation = NA,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

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

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

- If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
- A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
- A function will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A function can be created from a `formula` (e.g. `~ head(.x, 10)`).

**geom**  Use to override the default connection between `stat_lineribbon()` and `geom_lineribbon()`.

**position**  Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" (`position_dodge()`) or "dodgejust" (`position_dodgejust()`) can be useful if you have overlapping geometries.

**...**  Other arguments passed to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3` (see `Aesthetics`, below). They may also be parameters to the paired geom/stat. When paired with the default geom, `geom_lineribbon()`, these include:

**step**  Should the line/ribbon be drawn as a step function? One of:

- `FALSE` (default): do not draw as a step function.
- "mid" (or `TRUE`): draw steps midway between adjacent x values.
- "hv": draw horizontal-then-vertical steps.
- "vh": draw as vertical-then-horizontal steps.

`TRUE` is an alias for "mid" because for a step function with ribbons, "mid" is probably what you want (for the other two step approaches the ribbons at either the very first or very last x value will not be visible).
The `.width` argument passed to `point_interval`: a vector of probabilities to use that determine the widths of the resulting intervals. If multiple probabilities are provided, multiple intervals per group are generated, each with a different probability interval (and value of the corresponding `.width` and `level` generated variables).

`point_interval` A function from the `point_interval()` family (e.g., `median_qi`, `mean_qi`, `mode_hdi`, etc), or a string giving the name of a function from that family (e.g., "median_qi", "mean_qi", "mode_hdi", etc; if a string, the caller’s environment is searched for the function, followed by the `ggdist` environment). This function determines the point summary (typically mean, median, or mode) and interval type (quantile interval, qi; highest-density interval, hdi; or highest-density continuous interval, hdci). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of `orientation`. See the `point_interval()` family of functions for more information.

`orientation` Whether this geom is drawn horizontally or vertically. One of:

- `NA` (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
- "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify different groups. For each group, uses the x, xmin, xmax, and thickness aesthetics to draw points, intervals, and slabs.
- "vertical" (or "x"): draw vertically, using the x aesthetic to identify different groups. For each group, uses the y, ymin, ymax, and thickness aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for `orientation", "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal" (tidybayes had an orientation parameter before base ggplot did, hence the discrepancy).

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

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

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

**Details**

To visualize sample data, such as a data distribution, samples from a bootstrap distribution, or a Bayesian posterior, you can supply samples to the x or y aesthetic.

To visualize analytical distributions, you can use the `xdist` or `ydist` aesthetic. For historical reasons, you can also use `dist` to specify the distribution, though this is not recommended as it does not work as well with orientation detection. These aesthetics can be used as follows:

- `xdist`, `ydist`, and `dist` can be any distribution object from the `distributional` package (`dist_normal()`, `dist_beta()`, etc) or can be a `posterior::rvar()` object. Since these functions are vectorized, other columns can be passed directly to them in an `aes()` specification; e.g. `aes(dist = dist_normal(mu, sigma))` will work if mu and sigma are columns in the input data frame.
dist can be a character vector giving the distribution name. Then the arg1, ... arg9 aesthetics (or args as a list column) specify distribution arguments. Distribution names should correspond to R functions that have "p", "q", and "d" functions; e.g. "norm" is a valid distribution name because R defines the \texttt{pnorm()}, \texttt{qnorm()}, and \texttt{dnorm()} functions for Normal distributions.

See the \texttt{parse_dist()} function for a useful way to generate dist and args values from human-readable distribution specs (like \texttt{"normal(0,1)"}). Such specs are also produced by other packages (like the \texttt{brms::get_prior} function in brms); thus, \texttt{parse_dist()} combined with the stats described here can help you visualize the output of those functions.

Value

A \texttt{ggplot2::Stat} representing a line + multiple-ribbon geometry which can be added to a \texttt{ggplot()} object.

Computed Variables

The following variables are computed by this stat and made available for use in aesthetic specifications (\texttt{aes()}) using the \texttt{stat()} or \texttt{after_stat()} functions:

- x or y: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is x or y depends on orientation
- xmin or ymin: For intervals, the lower end of the interval from the interval function.
- xmax or ymax: For intervals, the upper end of the interval from the interval function.
- \texttt{.width}: For intervals, the interval width as a numeric value in [0, 1]. For slabs, the width of the smallest interval containing that value of the slab.
- level: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.
- pdf: For slabs, the probability density function (PDF). If \texttt{options("ggdist.experimental.slab_data_in_intervals"} is \texttt{TRUE}): For intervals, the PDF at the point summary; intervals also have \texttt{pdf_min} and \texttt{pdf_max} for the PDF at the lower and upper ends of the interval.
- cdf: For slabs, the cumulative distribution function. If \texttt{options("ggdist.experimental.slab_data_in_intervals"} is \texttt{TRUE}): For intervals, the CDF at the point summary; intervals also have \texttt{cdf_min} and \texttt{cdf_max} for the CDF at the lower and upper ends of the interval.

Aesthetics

The line+ribbon stats and geoms have a wide variety of aesthetics that control the appearance of their two sub-geometries: the line and the ribbon.

These stats support the following aesthetics:

- x: x position of the geometry (when orientation = "vertical"); or sample data to be summarized (when orientation = "horizontal" with sample data).
- y: y position of the geometry (when orientation = "horizontal"); or sample data to be summarized (when orientation = "vertical" with sample data).
- xdist: When using analytical distributions, distribution to map on the x axis: a \texttt{distributional} object (e.g. \texttt{dist_normal()}) or a \texttt{posterior::rvar()} object.
stat_lineribbon

- `ydist`: When using analytical distributions, distribution to map on the y axis: a `distributional` object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `dist`: When using analytical distributions, a name of a distribution (e.g. "norm"), a `distributional` object (e.g. `dist_normal()`), or a `posterior::rvar()` object. See Details.
- `args`: Distribution arguments (args or arg1, ... arg9). See Details.

In addition, in their default configuration (paired with `geom_lineribbon()`) the following aesthetics are supported by the underlying geom:

**Ribbon-specific aesthetics**

- `xmin`: Left edge of the ribbon sub-geometry (if orientation = "horizontal").
- `xmax`: Right edge of the ribbon sub-geometry (if orientation = "horizontal").
- `ymin`: Lower edge of the ribbon sub-geometry (if orientation = "vertical").
- `ymax`: Upper edge of the ribbon sub-geometry (if orientation = "vertical").

**Color aesthetics**

- `colour`: (or color) The color of the line sub-geometry.
- `fill`: The fill color of the ribbon sub-geometry.
- `alpha`: The opacity of the line and ribbon sub-geometries.
- `fill_ramp`: A secondary scale that modifies the fill scale to "ramp" to another color. See `scale_fill_ramp()` for examples.

**Line aesthetics**

- `size`: Width of line.
- `linetype`: Type of line (e.g., "solid", "dashed", etc)

**Other aesthetics** (these work as in standard geoms)

- `group`

See examples of some of these aesthetics in action in vignette("lineribbon"). Learn more about the sub-geom override aesthetics (like interval_color) in the scales documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").

**See Also**

See `geom_lineribbon()` for the geom underlying this stat.

Other lineribbon stats: `stat_ribbon()`
Example

```r
library(dplyr)
library(ggplot2)
library(distributional)

theme_set(theme_ggdist())

# ON SAMPLE DATA
tibble(x = 1:10) %>%
  group_by_all() %>%
do(tibble(y = rnorm(100, .x))) %>%
ggplot(aes(x = x, y = y)) +
  stat_lineribbon() +
  scale_fill_brewer()

# ON ANALYTICAL DISTRIBUTIONS
# Vectorized distribution types, like distributional::dist_normal()
# and posterior::rvar(), can be used with the `xdist` / `ydist` aesthetics
tibble(
  x = 1:10,
sd = seq(1, 3, length.out = 10)
) %>%
ggplot(aes(x = x, ydist = dist_normal(x, sd))) +
  stat_lineribbon() +
  scale_fill_brewer()
```

---

**stat_pointinterval**  
Point + multiple-interval plot (shortcut stat)

**Description**

Shortcut version of `stat_slabinterval()` with `geom_pointinterval()` for creating point + multiple-interval plots.

Roughly equivalent to:

```r
stat_slabinterval(
  geom = "pointinterval",
  show_slab = FALSE
)
```

**Usage**

```r
stat_pointinterval(
  mapping = NULL,
  data = NULL,
  geom = "pointinterval",
  position = "identity",
  ...,
)```
stat_pointinterval

```r
point_interval = "median_qi",
.width = c(0.66, 0.95),
orientation = NA,
nr.m = FALSE,
show.legend = c(size = FALSE),
inherit.aes = TRUE
```

**Arguments**

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

- **data**
  The data to be displayed in this layer. There are three options:
  - If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
  - A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
  - A function will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A function can be created from a `formula` (e.g. `~ head(.x, 10)`).

- **geom**
  Use to override the default connection between `stat_pointinterval()` and `geom_pointinterval()`

- **position**
  Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" (`position_dodge()`) or "dodgejust" (`position_dodgejust()`) can be useful if you have overlapping geometries.

- **...**
  Other arguments passed to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3` (see `Aesthetics`, below). They may also be parameters to the paired geom/stat. When paired with the default geom, `geom_pointinterval()`, these include:

  - **interval_size_domain**
    A length-2 numeric vector giving the minimum and maximum of the values of the size aesthetic that will be translated into actual sizes for intervals drawn according to `interval_size_range` (see the documentation for that argument.)

  - **interval_size_range**
    A length-2 numeric vector. This geom scales the raw size aesthetic values when drawing interval and point sizes, as they tend to be too thick when using the default settings of `scale_size_continuous()`, which give sizes with a range of `c(1, 6)`. The `interval_size_domain` value indicates the input domain of raw size values (typically this should be equal to the value of the range argument of the `scale_size_continuous()` function), and `interval_size_range` indicates the desired output range of the size values (the min and max of the actual sizes used to draw intervals). Most of the time it is not recommended to change the value of this argument, as it may result in strange scaling of legends; this argument is a holdover from earlier versions that did not have size aesthetics targeting the point and interval separately. If you want to adjust the size of the interval or
points separately, you can instead use the `interval_size` or `point_size` aesthetics; see `scales`.

`fatten_point` A multiplicative factor used to adjust the size of the point relative to the size of the thickest interval line. If you wish to specify point sizes directly, you can also use the `point_size` aesthetic and `scale_point_size_continuous()` or `scale_point_size_discrete()`: sizes specified with that aesthetic will not be adjusted using `fatten_point`.

`point_interval` A function from the `point_interval()` family (e.g., `median_qi`, `mean_qi`, `mode_hdi`, etc), or a string giving the name of a function from that family (e.g., "median_qi", "mean_qi", "mode_hdi", etc; if a string, the caller’s environment is searched for the function, followed by the `ggdist` environment). This function determines the point summary (typically mean, median, or mode) and interval type (quantile interval, qi; highest-density interval, hdi; or highest-density continuous interval, hdci). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of `orientation`. See the `point_interval()` family of functions for more information.

`.width` The `.width` argument passed to `point_interval`: a vector of probabilities to use that determine the widths of the resulting intervals. If multiple probabilities are provided, multiple intervals per group are generated, each with a different probability interval (and value of the corresponding `.width` and `level` generated variables).

`orientation` Whether this geom is drawn horizontally or vertically. One of:

- NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
- "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify different groups. For each group, uses the x, xmin, xmax, and thickness aesthetics to draw points, intervals, and slabs.
- "vertical" (or "x"): draw vertically, using the x aesthetic to identify different groups. For each group, uses the y, ymin, ymax, and thickness aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for `orientation`, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal" (tidybayes had an orientation parameter before base ggplot did, hence the discrepancy).

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

`show.legend` Should this layer be included in the legends? Default is `c(size = FALSE)`, unlike most geoms, to match its common use cases. FALSE hides all legends, TRUE shows all legends, and NA shows only those that are mapped (the default for most geoms).

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

To visualize sample data, such as a data distribution, samples from a bootstrap distribution, or a Bayesian posterior, you can supply samples to the x or y aesthetic.

To visualize analytical distributions, you can use the xdist or ydist aesthetic. For historical reasons, you can also use dist to specify the distribution, though this is not recommended as it does not work as well with orientation detection. These aesthetics can be used as follows:

- **xdist, ydist, and dist** can be any distribution object from the distributional package (dist_normal(), dist_beta(), etc) or can be a posterior::rvar() object. Since these functions are vectorized, other columns can be passed directly to them in an aes() specification; e.g. aes(dist = dist_normal(mu, sigma)) will work if mu and sigma are columns in the input data frame.

- **dist** can be a character vector giving the distribution name. Then the arg1, ... arg9 aesthetics (or args as a list column) specify distribution arguments. Distribution names should correspond to R functions that have "p", "q", and "d" functions; e.g. "norm" is a valid distribution name because R defines the pnorm(), qnorm(), and dnorm() functions for Normal distributions.

See the parse_dist() function for a useful way to generate dist and args values from human-readable distribution specs (like "normal(0,1)"). Such specs are also produced by other packages (like the brms::get_prior function in brms); thus, parse_dist() combined with the stats described here can help you visualize the output of those functions.

Value

A ggplot2::Stat representing a point + multiple-interval geometry which can be added to a ggplot() object.

Computed Variables

The following variables are computed by this stat and made available for use in aesthetic specifications (aes()) using the stat() or after_stat() functions:

- **x or y**: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is x or y depends on orientation.
- **xmin or ymin**: For intervals, the lower end of the interval from the interval function.
- **xmax or ymax**: For intervals, the upper end of the interval from the interval function.
- **.width**: For intervals, the interval width as a numeric value in [0, 1]. For slabs, the width of the smallest interval containing that value of the slab.
- **level**: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.
- **pdf**: For slabs, the probability density function (PDF). If options("ggdist.experimental.slab_data_in_intervals") is TRUE: For intervals, the PDF at the point summary; intervals also have pdf_min and pdf_max for the PDF at the lower and upper ends of the interval.
- **cdf**: For slabs, the cumulative distribution function. If options("ggdist.experimental.slab_data_in_intervals") is TRUE: For intervals, the CDF at the point summary; intervals also have cdf_min and cdf_max for the CDF at the lower and upper ends of the interval.
Aesthetics

The slab+interval stats and geoms have a wide variety of aesthetics that control the appearance of their three sub-geometries: the slab, the point, and the interval.

These stats support the following aesthetics:

- **x**: x position of the geometry (when orientation = "vertical"); or sample data to be summarized (when orientation = "horizontal" with sample data).
- **y**: y position of the geometry (when orientation = "horizontal"); or sample data to be summarized (when orientation = "vertical" with sample data).
- **xdist**: When using analytical distributions, distribution to map on the x axis: a distributional object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- **ydist**: When using analytical distributions, distribution to map on the y axis: a distributional object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- **dist**: When using analytical distributions, a name of a distribution (e.g. "norm"), a distributional object (e.g. `dist_normal()`) or a `posterior::rvar()` object. See Details.
- **args**: Distribution arguments (args or arg1, ... arg9). See Details.

In addition, in their default configuration (paired with `geom_pointinterval()`) the following aesthetics are supported by the underlying geom:

**Interval-specific aesthetics**

- **xmin**: Left end of the interval sub-geometry (if orientation = "horizontal").
- **xmax**: Right end of the interval sub-geometry (if orientation = "horizontal").
- **ymin**: Lower end of the interval sub-geometry (if orientation = "vertical").
- **ymax**: Upper end of the interval sub-geometry (if orientation = "vertical").

**Point-specific aesthetics**

- **shape**: Shape type used to draw the point sub-geometry.

**Color aesthetics**

- **colour**: (or color) The color of the interval and point sub-geometries. Use the slab_color, interval_color, or point_color aesthetics (below) to set sub-geometry colors separately.
- **fill**: The fill color of the slab and point sub-geometries. Use the slab_fill or point_fill aesthetics (below) to set sub-geometry colors separately.
- **alpha**: The opacity of the slab, interval, and point sub-geometries. Use the slab_alpha, interval_alpha, or point_alpha aesthetics (below) to set sub-geometry colors separately.
- **colour_ramp**: (or color_ramp) A secondary scale that modifies the color scale to "ramp" to another color. See `scale_colour_ramp()` for examples.
- **fill_ramp**: A secondary scale that modifies the fill scale to "ramp" to another color. See `scale_fill_ramp()` for examples.

**Line aesthetics**
• size: Width of the outline around the \texttt{slab} (if visible). Also determines the width of the line used to draw the \texttt{interval} and the size of the \texttt{point}, but raw size values are transformed according to the \texttt{interval_size_domain}, \texttt{interval_size_range}, and \texttt{fatten_point} parameters of the geom (see above). Use the \texttt{slab_size}, \texttt{interval_size}, or \texttt{point_size} aesthetics (below) to set sub-geometry line widths separately (note that when size is set directly using the override aesthetics, interval and point sizes are not affected by \texttt{interval_size_domain}, \texttt{interval_size_range}, and \texttt{fatten_point}).

• stroke: Width of the outline around the \texttt{point} sub-geometry.

• linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the \texttt{interval} and the outline of the \texttt{slab} (if it is visible). Use the \texttt{slab_linetype} or \texttt{interval_linetype} aesthetics (below) to set sub-geometry line types separately.

\textbf{Interval-specific color/line override aesthetics}

• \texttt{interval_colour}: (or \texttt{interval_color}) Override for \texttt{colour/color}: the color of the interval.

• \texttt{interval_alpha}: Override for \texttt{alpha}: the opacity of the interval.

• \texttt{interval_size}: Override for \texttt{size}: the line width of the interval.

• \texttt{interval_linetype}: Override for \texttt{linetype}: the line type of the interval.

\textbf{Point-specific color/line override aesthetics}

• \texttt{point_fill}: Override for \texttt{fill}: the fill color of the point.

• \texttt{point_colour}: (or \texttt{point_color}) Override for \texttt{colour/color}: the outline color of the point.

• \texttt{point_alpha}: Override for \texttt{alpha}: the opacity of the point.

• \texttt{point_size}: Override for \texttt{size}: the size of the point.

\textbf{Other aesthetics} (these work as in standard geoms)

• width

• height

• group

See examples of some of these aesthetics in action in vignette("slabinterval"). Learn more about the sub-geom override aesthetics (like \texttt{interval_color}) in the scales documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").

\textbf{See Also}

See \texttt{geom_pointinterval()} for the geom underlying this stat. See \texttt{stat_slabinterval()} for the stat this shortcut is based on.

Other slabinterval stats: \texttt{stat_ccdfinterval()}, \texttt{stat_cdfinterval()}, \texttt{stat_eye()}, \texttt{stat_gradientinterval()}, \texttt{stat_halfeye()}, \texttt{stat_histinterval()}, \texttt{stat_interval()}, \texttt{stat_slab()}

Examples

```r
# library(dplyr)
# library(ggplot2)
# library(distributional)

# theme_set(theme_ggdist())

# ON SAMPLE DATA
set.seed(1234)
df = data.frame(
    group = c("a", "b", "c"),
    value = rnorm(1500, mean = c(5, 7, 9), sd = c(1, 1.5, 1))
)
df %>%
    ggplot(aes(x = value, y = group)) +
    stat_pointinterval()

# ON ANALYTICAL DISTRIBUTIONS
dist_df = data.frame(
    group = c("a", "b", "c"),
    mean = c(5, 7, 8),
    sd = c(1, 1.5, 1)
)
# Vectorized distribution types, like distributional::dist_normal()
# and posterior::rvar(), can be used with the `xdist` / `ydist` aesthetics
dist_df %>%
    ggplot(aes(y = group, xdist = dist_normal(mean, sd))) +
    stat_pointinterval()
```

---

### stat_ribbon

**Multiple-ribbon plot (shortcut stat)**

#### Description

A combination of `stat_slabinterval()` and `geom_lineribbon()` with sensible defaults for making multiple-ribbon plots. While `geom_lineribbon()` is intended for use on data frames that have already been summarized using a `point_interval()` function, `stat_ribbon()` is intended for use directly on data frames of draws or of analytical distributions, and will perform the summarization using a `point_interval()` function.

Roughly equivalent to:

```r
stat_lineribbon(
    aes(colour = stat(I(NA)))
)
```
Usage

stat_ribbon(
    mapping = NULL,
    data = NULL,
    geom = "lineribbon",
    position = "identity",
    ...,
    .width = c(0.5, 0.8, 0.95),
    point_interval = "median_qi",
    orientation = NA,
    na.rm = FALSE,
    show.legend = NA,
    inherit.aes = TRUE
)

Arguments

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

data The data to be displayed in this layer. There are three options:
If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot().
A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.
A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).

geom Use to override the default connection between stat_ribbon() and geom_lineribbon()

position Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" (position_dodge()) or "dodgejust" (position_dodgejust()) can be useful if you have overlapping geometries.

... Other arguments passed to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3 (see Aesthetics, below). They may also be parameters to the paired geom/stat. When paired with the default geom, geom_lineribbon(), these include:

step Should the line/ribbon be drawn as a step function? One of:
• FALSE (default): do not draw as a step function.
• "mid" (or TRUE): draw steps midway between adjacent x values.
• "hv": draw horizontal-then-vertical steps.
• "vh": draw as vertical-then-horizontal steps.
TRUE is an alias for "mid" because for a step function with ribbons, "mid" is probably what you want (for the other two step approaches the ribbons at either the very first or very last x value will not be visible).
The `.width` argument passed to `point_interval`: a vector of probabilities to use that determine the widths of the resulting intervals. If multiple probabilities are provided, multiple intervals per group are generated, each with a different probability interval (and value of the corresponding `.width` and `level` generated variables).

`point_interval` A function from the `point_interval()` family (e.g., `median_qi`, `mean_qi`, `mode_hdi`, etc), or a string giving the name of a function from that family (e.g., "median_qi", "mean_qi", "mode_hdi", etc; if a string, the caller's environment is searched for the function, followed by the `ggdist` environment). This function determines the point summary (typically mean, median, or mode) and interval type (quantile interval, qi; highest-density interval, hdi; or highest-density continuous interval, hdci). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of `orientation`. See the `point_interval()` family of functions for more information.

`orientation` Whether this geom is drawn horizontally or vertically. One of:

- NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
- "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify different groups. For each group, uses the x, xmin, xmax, and thickness aesthetics to draw points, intervals, and slabs.
- "vertical" (or "x"): draw vertically, using the x aesthetic to identify different groups. For each group, uses the y, ymin, ymax, and thickness aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for `orientation`, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal" (tidybayes had an orientation parameter before base ggplot did, hence the discrepancy).

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

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

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

Details

To visualize sample data, such as a data distribution, samples from a bootstrap distribution, or a Bayesian posterior, you can supply samples to the x or y aesthetic.

To visualize analytical distributions, you can use the `xdist` or `ydist` aesthetic. For historical reasons, you can also use `dist` to specify the distribution, though this is not recommended as it does not work as well with orientation detection. These aesthetics can be used as follows:

- `xdist`, `ydist`, and `dist` can be any distribution object from the `distributional` package (`dist_normal()`, `dist_beta()`, etc) or can be a `posterior::rvar()` object. Since these functions are vectorized, other columns can be passed directly to them in an `aes()` specification; e.g. `aes(dist = dist_normal(mu, sigma))` will work if mu and sigma are columns in the input data frame.
dist can be a character vector giving the distribution name. Then the arg1, ... arg9 aesthetics (or args as a list column) specify distribution arguments. Distribution names should correspond to R functions that have “p”, “q”, and “d” functions; e.g. “norm” is a valid distribution name because R defines the pnorm(), qnorm(), and dnorm() functions for Normal distributions.

See the parse_dist() function for a useful way to generate dist and args values from human-readable distribution specs (like "normal(0,1)"). Such specs are also produced by other packages (like the brms::get_prior function in brms); thus, parse_dist() combined with the stats described here can help you visualize the output of those functions.

Value

A ggplot2::Stat representing a multiple-ribbon geometry which can be added to a ggplot() object.

Computed Variables

The following variables are computed by this stat and made available for use in aesthetic specifications (aes()) using the stat() or after_stat() functions:

• x or y: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is x or y depends on orientation
• xmin or ymin: For intervals, the lower end of the interval from the interval function.
• xmax or ymax: For intervals, the upper end of the interval from the interval function.
• .width: For intervals, the interval width as a numeric value in [0, 1]. For slabs, the width of the smallest interval containing that value of the slab.
• level: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.
• pdf: For slabs, the probability density function (PDF). If options("ggdist.experimental.slab_data_in_intervals") is TRUE: For intervals, the PDF at the point summary; intervals also have pdf_min and pdf_max for the PDF at the lower and upper ends of the interval.
• cdf: For slabs, the cumulative distribution function. If options("ggdist.experimental.slab_data_in_intervals") is TRUE: For intervals, the CDF at the point summary; intervals also have cdf_min and cdf_max for the CDF at the lower and upper ends of the interval.

Aesthetics

The line+ribbon stats and geoms have a wide variety of aesthetics that control the appearance of their two sub-geometries: the line and the ribbon.

These stats support the following aesthetics:

• x: x position of the geometry (when orientation = "vertical"); or sample data to be summarized (when orientation = "horizontal" with sample data).
• y: y position of the geometry (when orientation = "horizontal"); or sample data to be summarized (when orientation = "vertical" with sample data).
• xdist: When using analytical distributions, distribution to map on the x axis: a distributional object (e.g. dist_normal()) or a posterior::rvar() object.
• ydist: When using analytical distributions, distribution to map on the y axis: a **distributional** object (e.g. `dist_normal()`) or a **posterior::rvar()** object.

• dist: When using analytical distributions, a name of a distribution (e.g. "norm"), a **distributional** object (e.g. `dist_normal()`), or a **posterior::rvar()** object. See Details.

• args: Distribution arguments (args or arg1, ... arg9). See Details.

In addition, in their default configuration (paired with **geom_lineribbon()**) the following aesthetics are supported by the underlying geom:

**Ribbon-specific aesthetics**

• xmin: Left edge of the ribbon sub-geometry (if orientation = "horizontal").

• xmax: Right edge of the ribbon sub-geometry (if orientation = "horizontal").

• ymin: Lower edge of the ribbon sub-geometry (if orientation = "vertical").

• ymax: Upper edge of the ribbon sub-geometry (if orientation = "vertical").

**Color aesthetics**

• colour: (or color) The color of the **line** sub-geometry.

• fill: The fill color of the **ribbon** sub-geometry.

• alpha: The opacity of the **line** and **ribbon** sub-geometries.

• fill_ramp: A secondary scale that modifies the fill scale to "ramp" to another color. See `scale_fill_ramp()` for examples.

**Other aesthetics** (these work as in standard geoms)

• group

See examples of some of these aesthetics in action in vignette("lineribbon"). Learn more about the sub-geom override aesthetics (like interval_color) in the **scales** documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").

**See Also**

See **geom_lineribbon()** for the geom underlying this stat.

Other lineribbon stats: **stat_lineribbon()**

**Examples**

```r
glimpse(iris)
library(dplyr)
library(ggplot2)
library(distributional)
theme_set(theme_ggdist())

# ON SAMPLE DATA
tibble(x = 1:10) %>%
group_by_all() %>%
do(tibble(y = rnorm(100, .x))) %>%
```
\begin{verbatim}
  ggplot(aes(x = x, y = y)) +
  stat_ribbon() +
  scale_fill_brewer()

# ON ANALYTICAL DISTRIBUTIONS
# Vectorized distribution types, like distributional::dist_normal()
# and posterior::rvar(), can be used with the `xdist` / `ydist` aesthetics
  tibble(
    x = 1:10,
    sd = seq(1, 3, length.out = 10)
  ) %>%
  ggplot(aes(x = x, ydist = dist_normal(x, sd))) +
  stat_ribbon() +
  scale_fill_brewer()
\end{verbatim}

---

### stat_slab

**Slab (ridge) plot (shortcut stat)**

**Description**

Shortcut version of `stat_slabinterval()` with `geom_slab()` for creating slab (ridge) plots. Roughly equivalent to:

\begin{verbatim}
  stat_slabinterval(
    aes(size = NULL),
    geom = "slab",
    show.legend = NA
  )
\end{verbatim}

**Usage**

\begin{verbatim}
  stat_slab(
    mapping = NULL,
    data = NULL,
    geom = "slab",
    position = "identity",
    ...,
    slab_type = "pdf",
    p_limits = c(NA, NA),
    adjust = 1,
    trim = TRUE,
    expand = FALSE,
    breaks = "Sturges",
    outline_bars = FALSE,
    limits = NULL,
    n = 501,
    orientation = NA,
    na.rm = FALSE,
\end{verbatim}
Arguments

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

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

data The data to be displayed in this layer. There are three options:
If \texttt{NULL}, the default, the data is inherited from the plot data as specified in the call to \texttt{ggplot()}.
A \texttt{data.frame}, or other object, will override the plot data. All objects will be fortified to produce a data frame. See \texttt{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 \texttt{data.frame}, and will be used as the layer data. A function can be created from a formula (e.g. \texttt{~ head(.x, 10)}).

geom Use to override the default connection between \texttt{stat_slab()} and \texttt{geom_slab()}

position Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" (\texttt{position_dodge()}) or "dodgejust" (\texttt{position_dodgejust()}) can be useful if you have overlapping geometries.

... Other arguments passed to \texttt{layer()}. These are often aesthetics, used to set an aesthetic to a fixed value, like \texttt{colour = "red"} or \texttt{size = 3} (see \texttt{Aesthetics}, below). They may also be parameters to the paired geom/stat. When paired with the default geom, \texttt{geom_slab()}, these include:

normalize How to normalize heights of functions input to the thickness aesthetic. One of:
• "all": normalize so that the maximum height across all data is 1.
• "panels": normalize within panels so that the maximum height in each panel is 1.
• "xy": normalize within the x/y axis opposite the orientation of this geom so that the maximum height at each value of the opposite axis is 1.
• "groups": normalize within values of the opposite axis and within each group so that the maximum height in each group is 1.
• "none": values are taken as is with no normalization (this should probably only be used with functions whose values are in \([0,1]\), such as CDFs).

fill_type What type of fill to use when the fill color or alpha varies within a slab. One of:
• "segments": breaks up the slab geometry into segments for each unique combination of fill color and alpha value. This approach is supported by all graphics devices and works well for sharp cutoff values, but can give ugly results if a large number of unique fill colors are being used (as in gradients, like in \texttt{stat_gradientinterval()}).
• "gradient": a `grid::linearGradient()` is used to create a smooth gradient fill. This works well for large numbers of unique fill colors, but requires R >= 4.1 and is not yet supported on all graphics devices. As of this writing, the `png()` graphics device with `type = "cairo"`, the `svg()` device, the `pdf()` device, and the `ragg::agg_png()` devices are known to support this option. On R < 4.1, this option will fall back to `fill_type = "segment"` with a message.

• "auto": attempts to use `fill_type = "gradient"` if support for it can be auto-detected. On R >= 4.2, support for gradients can be auto-detected on some graphics devices; if support is not detected, this option will fall back to `fill_type = "segments"` (in case of a false negative, `fill_type = "gradient"` can be set explicitly). On R < 4.2, support for gradients cannot be auto-detected, so this will always fall back to `fill_type = "segments"`, in which case you can set `fill_type = "gradient"` explicitly if you are using a graphics device that support gradients.

`slab_type` The type of slab function to calculate: probability density (or mass) function ("pdf"), cumulative distribution function ("cdf"), or complementary CDF ("ccdf").

`p_limits` Probability limits (as a vector of size 2) used to determine the lower and upper limits of the slab. E.g., if this is `c(.001, .999)`, then a slab is drawn for the distribution from the quantile at `p = .001` to the quantile at `p = .999`. If the lower (respectively upper) limit is `NA`, then the lower (upper) limit will be the minimum (maximum) of the distribution’s support if it is finite, and `0.001` (`0.999`) if it is not finite. E.g., if `p_limits` is `c(NA, NA)` on a gamma distribution the effective value of `p_limits` would be `c(0, .999)` since the gamma distribution is defined on `(0, Inf)”; whereas on a normal distribution it would be equivalent to `c(.001, .999) since the normal distribution is defined on `(-Inf, Inf)".

`adjust` If `slab_type` is "pdf", bandwidth for the density estimator for sample data is adjusted by multiplying it by this value. See `density()` for more information.

`trim` For sample data, should the density estimate be trimmed to the range of the input data? Default `TRUE`.

`expand` For sample data, should the slab be expanded to the limits of the scale? Default `FALSE`. Can be length two to control expansion to the lower and upper limit respectively.

`breaks` If `slab_type` is "histogram", the breaks parameter that is passed to `hist()` to determine where to put breaks in the histogram (for sample data).

`outline_bars` For sample data (if `slab_type` is "histogram") and for discrete analytical distributions (whose slabs are drawn as histograms), determines if outlines in between the bars are drawn when the `slab_color` aesthetic is used. If `FALSE` (the default), the outline is drawn only along the tops of the bars; if `TRUE`, outlines in between bars are also drawn.

`limits` Manually-specified limits for the slab, as a vector of length two. These limits are combined with those computed based on `p_limits` as well as the limits defined by the scales of the plot to determine the limits used to draw the slab functions: these limits specify the maximal limits; i.e., if specified, the limits will not be wider than these (but may be narrower). Use `NA` to leave a limit alone; e.g.
limits = c(0, NA) will ensure that the lower limit does not go below 0, but let the upper limit be determined by either p_limits or the scale settings.

n
Number of points at which to evaluate the function that defines the slab.

orientation
Whether this geom is drawn horizontally or vertically. One of:

- NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
- "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify different groups. For each group, uses the x, xmin, xmax, and thickness aesthetics to draw points, intervals, and slabs.
- "vertical" (or "x"): draw vertically, using the x aesthetic to identify different groups. For each group, uses the y, ymin, ymax, and thickness aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for orientation, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal" (tidybayes had an orientation parameter before base ggplot did, hence the discrepancy).

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

show.legend
Should this layer be included in the legends? Default is c(size = FALSE), unlike most geoms, to match its common use cases. FALSE hides all legends, TRUE shows all legends, and NA shows only those that are mapped (the default for most geoms).

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

Details

To visualize sample data, such as a data distribution, samples from a bootstrap distribution, or a Bayesian posterior, you can supply samples to the x or y aesthetic.

To visualize analytical distributions, you can use the xdist or ydist aesthetic. For historical reasons, you can also use dist to specify the distribution, though this is not recommended as it does not work as well with orientation detection. These aesthetics can be used as follows:

- xdist, ydist, and dist can be any distribution object from the distributional package (dist_normal(), dist_beta(), etc) or can be a posterior::rvar() object. Since these functions are vectorized, other columns can be passed directly to them in an aes() specification; e.g. aes(dist = dist_normal(mu, sigma)) will work if mu and sigma are columns in the input data frame.
- dist can be a character vector giving the distribution name. Then the arg1, ..., arg9 aesthetics (or args as a list column) specify distribution arguments. Distribution names should correspond to R functions that have "p", "q", and "d" functions; e.g. "norm" is a valid distribution name because R defines the pnorm(), qnorm(), and dnorm() functions for Normal distributions.

See the parse_dist() function for a useful way to generate dist and args values from human-readable distribution specs (like "normal(0,1)"). Such specs are also produced by other packages (like the brms::get_prior function in brms); thus, parse_dist() combined with the stats described here can help you visualize the output of those functions.
Value

A `ggplot2::Stat` representing a slab (ridge) geometry which can be added to a `ggplot()` object.

Computed Variables

The following variables are computed by this stat and made available for use in aesthetic specifications (`aes()`) using the `stat()` or `after_stat()` functions:

- **x** or **y**: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is x or y depends on orientation.
- **xmin** or **ymin**: For intervals, the lower end of the interval from the interval function.
- **xmax** or **ymax**: For intervals, the upper end of the interval from the interval function.
- **.width**: For intervals, the interval width as a numeric value in [0, 1]. For slabs, the width of the smallest interval containing that value of the slab.
- **level**: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.
- **pdf**: For slabs, the probability density function (PDF). If `options("ggdist.experimental.slab_data_in_intervals")` is TRUE: For intervals, the PDF at the point summary; intervals also have pdf_min and pdf_max for the PDF at the lower and upper ends of the interval.
- **cdf**: For slabs, the cumulative distribution function. If `options("ggdist.experimental.slab_data_in_intervals")` is TRUE: For intervals, the CDF at the point summary; intervals also have cdf_min and cdf_max for the CDF at the lower and upper ends of the interval.
- **f**: For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by slab_type.
- **n**: For slabs, the number of data points summarized into that slab. If the slab was created from an analytical distribution via the xdist, ydist, or dist aesthetic, n will be Inf.

Aesthetics

The slab+interval stats and geoms have a wide variety of aesthetics that control the appearance of their three sub-geometries: the **slab**, the **point**, and the **interval**.

These stats support the following aesthetics:

- **x**: x position of the geometry (when orientation = "vertical"); or sample data to be summarized (when orientation = "horizontal" with sample data).
- **y**: y position of the geometry (when orientation = "horizontal"); or sample data to be summarized (when orientation = "vertical" with sample data).
- **xdist**: When using analytical distributions, distribution to map on the x axis: a distributional object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- **ydist**: When using analytical distributions, distribution to map on the y axis: a distributional object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- **dist**: When using analytical distributions, a name of a distribution (e.g. "norm"), a distributional object (e.g. `dist_normal()`), or a `posterior::rvar()` object. See Details.
- **args**: Distribution arguments (args or arg1, ... arg9). See Details.
In addition, in their default configuration (paired with `geom_slab()`) the following aesthetics are supported by the underlying geom:

**Slab-specific aesthetics**

- **thickness**: The thickness of the slab at each x value (if orientation = "horizontal") or y value (if orientation = "vertical") of the slab.
- **side**: Which side to place the slab on. "topright", "top", and "right" are synonyms which cause the slab to be drawn on the top or the right depending on if orientation is "horizontal" or "vertical". "bottomleft", "bottom", and "left" are synonyms which cause the slab to be drawn on the bottom or the left depending on if orientation is "horizontal" or "vertical". "topleft" causes the slab to be drawn on the top or the left, and "bottomright" causes the slab to be drawn on the bottom or the right. "both" draws the slab mirrored on both sides (as in a violin plot).
- **scale**: What proportion of the region allocated to this geom to use to draw the slab. If scale = 1, slabs that use the maximum range will just touch each other. Default is 0.9 to leave some space.
- **justification**: Justification of the interval relative to the slab, where 0 indicates bottom/left justification and 1 indicates top/right justification (depending on orientation). If justification is NULL (the default), then it is set automatically based on the value of side: when side is "top"/"right" justification is set to 0, when side is "bottom"/"left" justification is set to 1, and when side is "both" justification is set to 0.5.
- **datatype**: When using composite geoms directly without a stat (e.g. `geom_slabinterval()`), datatype is used to indicate which part of the geom a row in the data targets: rows with datatype = "slab" target the slab portion of the geometry and rows with datatype = "interval" target the interval portion of the geometry. This is set automatically when using ggdist stats.

**Color aesthetics**

- **colour**: (or color) The color of the interval and point sub-geometries. Use the slab_color, interval_color, or point_color aesthetics (below) to set sub-geometry colors separately.
- **fill**: The fill color of the slab and point sub-geometries. Use the slab_fill or point_fill aesthetics (below) to set sub-geometry colors separately.
- **alpha**: The opacity of the slab, interval, and point sub-geometries. Use the slab_alpha, interval_alpha, or point_alpha aesthetics (below) to set sub-geometry colors separately.
- **colour_ramp**: (or color_ramp) A secondary scale that modifies the color scale to "ramp" to another color. See `scale_colour_ramp()` for examples.
- **fill_ramp**: A secondary scale that modifies the fill scale to "ramp" to another color. See `scale_fill_ramp()` for examples.

**Line aesthetics**

- **size**: Width of the outline around the slab (if visible). Also determines the width of the line used to draw the interval and the size of the point, but raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the slab_size, interval_size, or point_size aesthetics (below) to set sub-geometry line widths separately (note that when size is set directly using the override aesthetics, interval and point sizes are not affected by interval_size_domain, interval_size_range, and fatten_point).
• **stroke**: Width of the outline around the **point** sub-geometry.
• **linetype**: Type of line (e.g., "solid", "dashed", etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the **slab_linetype** or **interval_linetype** aesthetics (below) to set sub-geometry line types separately.

**Slab-specific color/line override aesthetics**
• **slab_fill**: Override for fill: the fill color of the slab.
• **slab_colour**: (or **slab_color**) Override for colour/color: the outline color of the slab.
• **slab_alpha**: Override for alpha: the opacity of the slab.
• **slab_size**: Override for size: the width of the outline of the slab.
• **slab_linetype**: Override for linetype: the line type of the outline of the slab.

**Other aesthetics** (these work as in standard geoms)
• **width**
• **height**
• **group**

See examples of some of these aesthetics in action in vignette("slabinterval"). Learn more about the sub-geom override aesthetics (like interval_color) in the **scales** documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").

**See Also**
See **geom_slab()** for the geom underlying this stat. See **stat_slabinterval()** for the stat this shortcut is based on.

Other slabinterval stats: **stat_ccdfinterval()**, **stat_cdfinterval()**, **stat_eye()**, **stat_gradientinterval()**, **stat_halfeye()**, **stat_histinterval()**, **stat_interval()**, **stat_pointinterval()**

**Examples**
```r
library(dplyr)
library(ggplot2)
library(distributional)

theme_set(theme_ggdist())

# ON SAMPLE DATA
set.seed(1234)
df = data.frame(
  group = c("a", "b", "c"),
  value = rnorm(1500, mean = c(5, 7, 9), sd = c(1, 1.5, 1))
)
df %>%
ggplot(aes(x = value, y = group)) +
  stat_slab()

# ON ANALYTICAL DISTRIBUTIONS
```
dist_df <- data.frame(
  group = c("a", "b", "c"),
  mean = c( 5, 7, 8),
  sd = c( 1, 1.5, 1)
)
# Vectorized distribution types, like distributional::dist_normal()
# and posterior::rvar(), can be used with the `xdist` / `ydist` aesthetics
dist_df %>%
  ggplot(aes(y = group, xdist = dist_normal(mean, sd))) +
  stat_slab()

# RIDGE PLOTS
# "ridge" plots can be created by expanding the slabs to the limits of the plot
# (expand = TRUE), allowing the density estimator to be nonzero outside the
# limits of the data (trim = FALSE), and increasing the height of the slabs.
data.frame(
  group = letters[1:3],
  value = rnorm(3000, 3:1)
)
) %>%
  ggplot(aes(y = group, x = value)) +
  stat_slab(color = "black", expand = TRUE, trim = FALSE, height = 2)

stat_slabinterval

**Slab + interval plots for sample data and analytical distributions (ggplot stat)**

**Description**

"Meta" stat for computing distribution functions (densities or CDFs) + intervals for use with `geom_slabinterval()`. Useful for creating eye plots, half-eye plots, CCDF bar plots, gradient plots, histograms, and more. Sample data can be supplied to the x and y aesthetics or analytical distributions (in a variety of formats) can be supplied to the xdist and ydist aesthetics. See **Details**.

**Usage**

stat_slabinterval(
  mapping = NULL,
  data = NULL,
  geom = "slabinterval",
  position = "identity",
  ...
  ,
  slab_type = "pdf",
  p_limits = c(NA, NA),
  adjust = 1,
  trim = TRUE,
  expand = FALSE,
  breaks = "Sturges",
  outline_bars = FALSE,
  point_interval = "median_qi",
  ...)
stat_slabinterval

limits = NULL,
 n = 501,
.width = c(0.66, 0.95),
orientation = NA,
na.rm = FALSE,
show.legend = c(size = FALSE),
inherit.aes = TRUE
)

Arguments

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

data The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.
A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).

gem Use to override the default connection between stat_slabinterval() and geom_slabinterval()

position Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" (position_dodge()) or "dodgejust" (position_dodgejust()) can be useful if you have overlapping geometries.

... Other arguments passed to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3 (see Aesthetics, below). They may also be parameters to the paired geom/stat. When paired with the default geom, geom_slabinterval(), these include:

normalize How to normalize heights of functions input to the thickness aesthetic. One of:
- "all": normalize so that the maximum height across all data is 1.
- "panels": normalize within panels so that the maximum height in each panel is 1.
- "xy": normalize within the x/y axis opposite the orientation of this geom so that the maximum height at each value of the opposite axis is 1.
- "groups": normalize within values of the opposite axis and within each group so that the maximum height in each group is 1.
- "none": values are taken as is with no normalization (this should probably only be used with functions whose values are in [0,1], such as CDFs).
fill_type What type of fill to use when the fill color or alpha varies within a slab. One of:

- "segments": breaks up the slab geometry into segments for each unique combination of fill color and alpha value. This approach is supported by all graphics devices and works well for sharp cutoff values, but can give ugly results if a large number of unique fill colors are being used (as in gradients, like in `stat_gradientinterval()`).
- "gradient": a `grid::linearGradient()` is used to create a smooth gradient fill. This works well for large numbers of unique fill colors, but requires R >= 4.1 and is not yet supported on all graphics devices. As of this writing, the `png()` graphics device with type = "cairo", the `svg()` device, the `pdf()` device, and the `ragg::agg_png()` devices are known to support this option. On R < 4.1, this option will fall back to `fill_type = "segment"` with a message.
- "auto": attempts to use `fill_type = "gradient"` if support for it can be auto-detected. On R >= 4.2, support for gradients can be auto-detected on some graphics devices; if support is not detected, this option will fall back to `fill_type = "segments"` (in case of a false negative, `fill_type = "gradient"` can be set explicitly). On R < 4.2, support for gradients cannot be auto-detected, so this will always fall back to `fill_type = "segments"`, in which case you can set `fill_type = "gradient"` explicitly if you are using a graphics device that support gradients.

interval_size_domain A length-2 numeric vector giving the minimum and maximum of the values of the size aesthetic that will be translated into actual sizes for intervals drawn according to `interval_size_range` (see the documentation for that argument.)

interval_size_range A length-2 numeric vector. This geom scales the raw size aesthetic values when drawing interval and point sizes, as they tend to be too thick when using the default settings of `scale_size_continuous()`, which give sizes with a range of c(1, 6). The `interval_size_domain` value indicates the input domain of raw size values (typically this should be equal to the value of the range argument of the `scale_size_continuous()` function), and `interval_size_range` indicates the desired output range of the size values (the min and max of the actual sizes used to draw intervals). Most of the time it is not recommended to change the value of this argument, as it may result in strange scaling of legends; this argument is a holdover from earlier versions that did not have size aesthetics targeting the point and interval separately. If you want to adjust the size of the interval or points separately, you can instead use the `interval_size` or `point_size` aesthetics; see `scales`.

fatten_point A multiplicative factor used to adjust the size of the point relative to the size of the thickest interval line. If you wish to specify point sizes directly, you can also use the `point_size` aesthetic and `scale_point_size_continuous()` or `scale_point_size_discrete()`: sizes specified with that aesthetic will not be adjusted using `fatten_point`.

slab_type The type of slab function to calculate: probability density (or mass) function ("pdf"), cumulative distribution function ("cdf"), or complementary CDF ("ccdf").
**stat_slabinterval**

- **p_limits**
  Probability limits (as a vector of size 2) used to determine the lower and upper limits of the slab. E.g., if this is `c(.001, .999)`, then a slab is drawn for the distribution from the quantile at \( p = .001 \) to the quantile at \( p = .999 \). If the lower (respectively upper) limit is \( \text{NA} \), then the lower (upper) limit will be the minimum (maximum) of the distribution’s support if it is finite, and 0.001 (0.999) if it is not finite. E.g., if \( \text{p_limits} \) is `c(\( \text{NA} \), \( \text{NA} \))` on a gamma distribution the effective value of \( \text{p_limits} \) would be `c(0, .999)` since the gamma distribution is defined on \((0, \infty)\); whereas on a normal distribution it would be equivalent to `c(.001, .999)` since the normal distribution is defined on \((-\infty, \infty)\).

- **adjust**
  If \( \text{slab_type} \) is "pdf", bandwidth for the density estimator for sample data is adjusted by multiplying it by this value. See `density()` for more information.

- **trim**
  For sample data, should the density estimate be trimmed to the range of the input data? Default **TRUE**.

- **expand**
  For sample data, should the slab be expanded to the limits of the scale? Default **FALSE**. Can be length two to control expansion to the lower and upper limit respectively.

- **breaks**
  If \( \text{slab_type} \) is "histogram", the breaks parameter that is passed to `hist()` to determine where to put breaks in the histogram (for sample data).

- **outline_bars**
  For sample data (if \( \text{slab_type} \) is "histogram") and for discrete analytical distributions (whose slabs are drawn as histograms), determines if outlines in between the bars are drawn when the \( \text{slab_color} \) aesthetic is used. If **FALSE** (the default), the outline is drawn only along the tops of the bars; if **TRUE**, outlines in between bars are also drawn.

- **point_interval**
  A function from the `point_interval()` family (e.g., `median_qi`, `mean_qi`, `mode_hdi`, etc), or a string giving the name of a function from that family (e.g., "median_qi", "mean_qi", "mode_hdi", etc; if a string, the caller’s environment is searched for the function, followed by the `ggdist` environment). This function determines the point summary (typically mean, median, or mode) and interval type (quantile interval, qi; highest-density interval, hdi; or highest-density continuous interval, hdci). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of `orientation`. See the `point_interval()` family of functions for more information.

- **limits**
  Manually-specified limits for the slab, as a vector of length two. These limits are combined with those computed based on \( \text{p_limits} \) as well as the limits defined by the scales of the plot to determine the limits used to draw the slab functions: these limits specify the maximal limits; i.e., if specified, the limits will not be wider than these (but may be narrower). Use \( \text{NA} \) to leave a limit alone; e.g. \( \text{limits} = c(\theta, \text{NA}) \) will ensure that the lower limit does not go below 0, but let the upper limit be determined by either \( \text{p_limits} \) or the scale settings.

- **n**
  Number of points at which to evaluate the function that defines the slab.

- **.width**
  The \( .width \) argument passed to `point_interval`: a vector of probabilities to use that determine the widths of the resulting intervals. If multiple probabilities are provided, multiple intervals per group are generated, each with a different probability interval (and value of the corresponding \( .width \) and \( \text{level} \) generated variables).

- **orientation**
  Whether this geom is drawn horizontally or vertically. One of:
• NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
• "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify different groups. For each group, uses the x, xmin, xmax, and thickness aesthetics to draw points, intervals, and slabs.
• "vertical" (or "x"): draw vertically, using the x aesthetic to identify different groups. For each group, uses the y, ymin, ymax, and thickness aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for orientation, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal" (tidybayes had an orientation parameter before base ggplot did, hence the discrepancy).

na.rm If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
show.legend Should this layer be included in the legends? Default is c(size = FALSE), unlike most geoms, to match its common use cases. FALSE hides all legends, TRUE shows all legends, and NA shows only those that are mapped (the default for most geoms).
inherit.aes If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. borders().

Details

A highly configurable stat for generating a variety of plots that combine a "slab" that describes a distribution plus a point summary and any number of intervals. Several "shortcut" stats are provided which combine multiple options to create useful geoms, particularly eye plots (a violin plot of density plus interval), half-eye plots (a density plot plus interval), CCDF bar plots (a complementary CDF plus interval), and gradient plots (a density encoded in color alpha plus interval).

The shortcut stats include:
• stat_eye(): Eye plots (violin + interval)
• stat_halfeye(): Half-eye plots (density + interval)
• stat_ccdfinterval(): CCDF bar plots (CCDF + interval)
• stat_cdfinterval(): CDF bar plots (CDF + interval)
• stat_gradientinterval(): Density gradient + interval plots
• stat_slab(): Density plots
• stat_histinterval(): Histogram + interval plots
• stat_pointinterval(): Point + interval plots
• stat_interval(): Interval plots

To visualize sample data, such as a data distribution, samples from a bootstrap distribution, or a Bayesian posterior, you can supply samples to the x or y aesthetic.

To visualize analytical distributions, you can use the xdist or ydist aesthetic. For historical reasons, you can also use dist to specify the distribution, though this is not recommended as it does not work as well with orientation detection. These aesthetics can be used as follows:
• `xdist`, `ydist`, and `dist` can be any distribution object from the `distributional` package (`dist_normal()`, `dist_beta()`, etc) or can be a `posterior::rvar()` object. Since these functions are vectorized, other columns can be passed directly to them in an `aes()` specification; e.g. `aes(dist = dist_normal(mu, sigma))` will work if `mu` and `sigma` are columns in the input data frame.

• `dist` can be a character vector giving the distribution name. Then the `arg1, … arg9` aesthetics (or `args` as a list column) specify distribution arguments. Distribution names should correspond to R functions that have “p”, “q”, and “d” functions; e.g. "norm" is a valid distribution name because R defines the `pnorm()`, `qnorm()`, and `dnorm()` functions for Normal distributions.

See the `parse_dist()` function for a useful way to generate `dist` and `args` values from human-readable distribution specs (like "normal(0,1)"). Such specs are also produced by other packages (like the `brms::get_prior` function in `brms`); thus, `parse_dist()` combined with the stats described here can help you visualize the output of those functions.

Value

A `ggplot2::Stat` representing a slab or combined slab+interval geometry which can be added to a `ggplot()` object.

Computed Variables

The following variables are computed by this stat and made available for use in aesthetic specifications (`aes()`) using the `stat()` or `after_stat()` functions:

• `x` or `y`: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is `x` or `y` depends on `orientation`

• `xmin` or `ymin`: For intervals, the lower end of the interval from the interval function.

• `xmax` or `ymax`: For intervals, the upper end of the interval from the interval function.

• `.width`: For intervals, the interval width as a numeric value in `[0, 1]`. For slabs, the width of the smallest interval containing that value of the slab.

• `level`: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.

• `pdf`: For slabs, the probability density function (PDF). If `options("ggdist.experimental.slab_data_in_intervals")` is `TRUE`: For intervals, the PDF at the point summary; intervals also have `pdf_min` and `pdf_max` for the PDF at the lower and upper ends of the interval.

• `cdf`: For slabs, the cumulative distribution function. If `options("ggdist.experimental.slab_data_in_intervals")` is `TRUE`: For intervals, the CDF at the point summary; intervals also have `cdf_min` and `cdf_max` for the CDF at the lower and upper ends of the interval.

• `f`: For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by `slab_type`.

• `n`: For slabs, the number of data points summarized into that slab. If the slab was created from an analytical distribution via the `xdist`, `ydist`, or `dist` aesthetic, `n` will be `Inf`. 
Aesthetics

The slab+interval stats and geoms have a wide variety of aesthetics that control the appearance of their three sub-geometries: the slab, the point, and the interval.

These stats support the following aesthetics:

- \( x \): x position of the geometry (when orientation = "vertical"); or sample data to be summarized (when orientation = "horizontal" with sample data).
- \( y \): y position of the geometry (when orientation = "horizontal"); or sample data to be summarized (when orientation = "vertical" with sample data).
- \( xdist \): When using analytical distributions, distribution to map on the x axis: a distributional object (e.g. \( \text{dist\_normal()} \)) or a \( \text{posterior::rvar()} \) object.
- \( ydist \): When using analytical distributions, distribution to map on the y axis: a distributional object (e.g. \( \text{dist\_normal()} \)) or a \( \text{posterior::rvar()} \) object.
- \( dist \): When using analytical distributions, a name of a distribution (e.g. "norm"), a distributional object (e.g. \( \text{dist\_normal()} \)), or a \( \text{posterior::rvar()} \) object. See Details.
- \( args \): Distribution arguments (\( args \) or \( \text{arg1, \ldots, arg9} \)). See Details.

In addition, in their default configuration (paired with \( \text{geom\_slabinterval()} \)) the following aesthetics are supported by the underlying geom:

**Slab-specific aesthetics**

- \( \text{thickness} \): The thickness of the slab at each x value (if orientation = "horizontal") or y value (if orientation = "vertical") of the slab.
- \( \text{side} \): Which side to place the slab on. "topright", "top", and "right" are synonyms which cause the slab to be drawn on the top or the right depending on if orientation is "horizontal" or "vertical". "bottomleft", "bottom", and "left" are synonyms which cause the slab to be drawn on the bottom or the left depending on if orientation is "horizontal" or "vertical". "topleft" causes the slab to be drawn on the top and the left, and "bottomright" causes the slab to be drawn on the bottom or the right. "both" draws the slab mirrored on both sides (as in a violin plot).
- \( \text{scale} \): What proportion of the region allocated to this geom to use to draw the slab. If scale = 1, slabs that use the maximum range will just touch each other. Default is 0.9 to leave some space.
- \( \text{justification} \): Justification of the interval relative to the slab, where 0 indicates bottom/left justification and 1 indicates top/right justification (depending on orientation). If justification is NULL (the default), then it is set automatically based on the value of side: when side is "top"/"right" justification is set to 0, when side is "bottom"/"left" justification is set to 1, and when side is "both" justification is set to 0.5.
- \( \text{datatype} \): When using composite geoms directly without a stat (e.g. \( \text{geom\_slabinterval()} \)), datatype is used to indicate which part of the geom a row in the data targets: rows with \( \text{datatype} = "slab" \) target the slab portion of the geometry and rows with \( \text{datatype} = "interval" \) target the interval portion of the geometry. This is set automatically when using ggdist stats.

**Interval-specific aesthetics**

- \( \text{xmin} \): Left end of the interval sub-geometry (if orientation = "horizontal").
• xmax: Right end of the interval sub-geometry (if orientation = "horizontal").
• ymin: Lower end of the interval sub-geometry (if orientation = "vertical").
• ymax: Upper end of the interval sub-geometry (if orientation = "vertical").

**Point-specific aesthetics**

• shape: Shape type used to draw the point sub-geometry.

**Color aesthetics**

• colour: (or color) The color of the interval and point sub-geometries. Use the slab_color, interval_color, or point_color aesthetics (below) to set sub-geometry colors separately.
• fill: The fill color of the slab and point sub-geometries. Use the slab_fill or point_fill aesthetics (below) to set sub-geometry colors separately.
• alpha: The opacity of the slab, interval, and point sub-geometries. Use the slab_alpha, interval_alpha, or point_alpha aesthetics (below) to set sub-geometry colors separately.
• colour_ramp: (or color_ramp) A secondary scale that modifies the color scale to "ramp" to another color. See scale_colour_ramp() for examples.
• fill_ramp: A secondary scale that modifies the fill scale to "ramp" to another color. See scale_fill_ramp() for examples.

**Line aesthetics**

• size: Width of the outline around the slab (if visible). Also determines the width of the line used to draw the interval and the size of the point, but raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the slab_size, interval_size, or point_size aesthetics (below) to set sub-geometry line widths separately (note that when size is set directly using the override aesthetics, interval and point sizes are not affected by interval_size_domain, interval_size_range, and fatten_point).
• stroke: Width of the outline around the point sub-geometry.
• linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the interval and the outline of the slab (if it is visible). Use the slab_linetype or interval_linetype aesthetics (below) to set sub-geometry line types separately.

**Slab-specific color/line override aesthetics**

• slab_fill: Override for fill: the fill color of the slab.
• slab_colour: (or slab_color) Override for colour/color: the outline color of the slab.
• slab_alpha: Override for alpha: the opacity of the slab.
• slab_size: Override for size: the width of the outline of the slab.
• slab_linetype: Override for linetype: the line type of the outline of the slab.

**Interval-specific color/line override aesthetics**

• interval_colour: (or interval_color) Override for colour/color: the color of the interval.
• interval_alpha: Override for alpha: the opacity of the interval.
• interval_size: Override for size: the line width of the interval.
• interval_linetype: Override for linetype: the line type of the interval.

**Point-specific color/line override aesthetics**

• point_fill: Override for fill: the fill color of the point.
• point_colour: (or point_color) Override for colour/color: the outline color of the point.
• point_alpha: Override for alpha: the opacity of the point.
• point_size: Override for size: the size of the point.

**Other aesthetics** (these work as in standard geoms)

• width
• height
• group

See examples of some of these aesthetics in action in vignette("slabinterval"). Learn more about the sub-geom override aesthetics (like interval_color) in the scales documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").

**See Also**

See `geom_slabinterval()` for more information on the geom these stats use by default and some of the options it has. See vignette("slabinterval") for a variety of examples of use.

**Examples**

```r
library(dplyr)
library(ggplot2)
library(distributional)

theme_set(theme_ggdist())

# EXAMPLES ON SAMPLE DATA
set.seed(1234)
df = data.frame(
  group = c("a", "b", "c", "c", "c"),
  value = rnorm(2500, mean = c(5, 7, 9, 9, 9), sd = c(1, 1.5, 1, 1, 1))
)

# here are vertical eyes:
df %>%
ggplot(aes(x = group, y = value)) +
stat_eye()

# note the sample size is not automatically incorporated into the
# area of the densities in case one wishes to plot densities against
```
# a reference (e.g. a prior distribution).
# But you may wish to account for sample size if using these geoms
# for something other than visualizing posteriors; in which case
# you can use stat(f*n):

```r
df %>%
ggplot(aes(x = group, y = value)) +
  stat_eye(aes(thickness = stat(pdf*n)))
```

# EXAMPLES ON ANALYTICAL DISTRIBUTIONS

```r
dist_df = tribble(~group, ~subgroup, ~mean, ~sd,
                   "a", "h", 5, 1,
                   "b", "h", 7, 1.5,
                   "c", "h", 8, 1,
                   "c", "i", 9, 1,
                   "c", "j", 7, 1)

dist_df %>%
ggplot(aes(x = group, ydist = dist_normal(mean, sd), fill = subgroup)) +
  stat_eye(position = "dodge")
```

# using the old character vector + args approach
```r
dist_df %>%
ggplot(aes(x = group, dist = "norm", arg1 = mean, arg2 = sd, fill = subgroup)) +
  stat_eye(position = "dodge")
```

# the stat_slabinterval family applies a Jacobian adjustment to densities
# when plotting on transformed scales in order to plot them correctly.
# It determines the Jacobian using symbolic differentiation if possible,
# using stats::D(). If symbolic differentiation fails, it falls back
# to numericDeriv(), which is less reliable; therefore, it is
# advisable to use scale transformation functions that are defined in
# terms of basic math functions so that their derivatives can be
# determined analytically (most of the transformation functions in the
# scales package currently have this property).
# For example, here is a log-Normal distribution plotted on the log
# scale, where it will appear Normal:
```r
data.frame(dist = "lnorm", logmean = log(10), logsd = 2*log(10)) %>%
ggplot(aes(y = 1, dist = dist, arg1 = logmean, arg2 = logsd)) +
  stat_halfeye() +
  scale_x_log10(breaks = 10^seq(-5,7, by = 2))
```

# see vignette("slabinterval") for many more examples.
 student_t

Scaled and shifted Student’s t distribution

Description

Density, distribution function, quantile function and random generation for the scaled and shifted Student’s t distribution, parameterized by degrees of freedom (df), location (mu), and scale (sigma).

Usage

dstudent_t(x, df, mu = 0, sigma = 1, log = FALSE)
pstudent_t(q, df, mu = 0, sigma = 1, lower.tail = TRUE, log.p = FALSE)
qstudent_t(p, df, mu = 0, sigma = 1, lower.tail = TRUE, log.p = FALSE)
rintstudent_t(n, df, mu = 0, sigma = 1)

Arguments

x, q vector of quantiles.
df degrees of freedom (> 0, maybe non-integer). df = Inf is allowed.
mu Location parameter (median)
sigma Scale parameter
log, log.p logical; if TRUE, probabilities p are given as log(p).
lower.tail logical; if TRUE (default), probabilities are \( P[X \leq x] \), otherwise, \( P[X > x] \).
p vector of probabilities.
n number of observations. If length(n) > 1, the length is taken to be the number required.

Value

- dstudent_t gives the density
- pstudent_t gives the cumulative distribution function (CDF)
- qstudent_t gives the quantile function (inverse CDF)
- rstudent_t generates random draws.

The length of the result is determined by n for rstudent_t, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

See Also

parse_dist() and parsing distribution specs and the stat_slabinterval() family of stats for visualizing them.
Examples

library(dplyr)
library(ggplot2)
library(forcats)

expand.grid(
  df = c(3, 5, 10, 30),
  scale = c(1, 1.5)
) %>%
ggplot(aes(y = 0, dist = "student_t", arg1 = df, arg2 = 0, arg3 = scale, color = ordered(df))) +
  stat_slab(p_limits = c(.01, .99), fill = NA) +
  scale_y_continuous(breaks = NULL) +
  facet_grid(~ scale) +
  labs(
    title = "dstudent_t(x, df, 0, sigma)",
    subtitle = "Scale (sigma)",
    y = NULL,
    x = NULL
  ) +
  theme_ggdist() +
  theme(axis.title = element_text(hjust = 0))

theme_ggdist

Simple, light ggplot2 theme for ggdist and tidybayes

Description

A simple, relatively minimalist ggplot2 theme, and some helper functions to go with it.

Usage

theme_ggdist()
theme_tidybayes()
facet_title_horizontal()
axis_titles_bottom_left()
facet_title_left_horizontal()
facet_title_right_horizontal()
Details

This is a relatively minimalist ggplot2 theme, intended to be used for making publication-ready plots. It is currently based on `ggplot2::theme_light()`.

A word of warning: this theme may (and very likely will) change in the future as I tweak it to my taste.

`theme_ggdist()` and `theme_tidybayes()` are aliases.

Value

A named list in the format of `ggplot2::theme()`

Author(s)

Matthew Kay

See Also

`ggplot2::theme()`, `ggplot2::theme_set()`

Examples

```r
library(ggplot2)
theme_set(theme_ggdist())
```

Description

These functions translate ggdist/tidybayes-style data frames to/from different data frame formats (each format using a different naming scheme for its columns).

Usage

```r
to_broom_names(data)
from_broom_names(data)
to_ggmcmc_names(data)
from_ggmcmc_names(data)
```
Arguments

data A data frame to translate.

Details

Function prefixed with `to_` translate from the ggdist/tidybayes format to another format, functions prefixed with `from_` translate from that format back to the ggdist/tidybayes format. Formats include:

- **to_broom_names() / from_broom_names()**:
  - `.variable` <-> `term`
  - `.value` <-> `estimate`
  - `.prediction` <-> `.fitted`
  - `.lower` <-> `conf.low`
  - `.upper` <-> `conf.high`

- **to_ggmcmc_names() / from_ggmcmc_names()**:
  - `.chain` <-> `Chain`
  - `.iteration` <-> `Iteration`
  - `.variable` <-> `Parameter`
  - `.value` <-> `value`

Value

A data frame with (possibly) new names in some columns, according to the translation scheme described in **Details**.

Author(s)

Matthew Kay

Examples

```r
library(dplyr)

data(RankCorr_u_tau, package = "ggdist")

df = RankCorr_u_tau %>%
  dplyr::rename(.variable = i, .value = u_tau) %>%
  group_by(.variable) %>%
  median_qi(.value)

df %>%
to_broom_names()
```
Index

* datasets
  ggdist-deprecated, 45
* ggdist scales
  scale_colour_ramp, 66
  scale_thickness, 69
  scales, 62
* lineribbon stats
  stat_lineribbon, 127
  stat_ribbon, 138
* manip
  tidy-format-translators, 162
* slabinterval geoms
  geom_interval, 21
  geom_pointinterval, 29
  geom_slab, 34
* slabinterval stats
  stat_ccdfinterval, 71
  stat_cdfinterval, 80
  stat_eye, 88
  stat_gradientinterval, 97
  stat_halfeye, 105
  stat_histinterval, 113
  stat_interval, 122
  stat_pointinterval, 132
  stat_slab, 143

aes(), 14, 17, 22, 26, 30, 34, 39, 72, 76, 81, 84, 89, 92, 93, 98, 101, 106, 109, 110, 114, 118, 122, 124, 128–130, 133, 135, 139–141, 144, 146, 147, 151, 155

aes_(), 14, 22, 26, 30, 34, 39, 72, 81, 89, 98, 106, 114, 122, 128, 133, 139, 144, 151

after_stat(), 76, 84, 93, 101, 110, 118, 124, 130, 135, 141, 147, 155

axis_titles_bottom_left (theme_ggdist), 161

beeswarm::beeswarm(), 4, 15

bin_dots, 4
bin_dots(), 11
borders(), 16, 23, 27, 31, 36, 41, 75, 84, 92, 101, 109, 117, 124, 129, 134, 140, 146, 154
cdf(), 9
continuous_scale(), 67, 70
coord_cartesian(), 65, 68, 70
curve_interval, 6
curve_interval(), 6
cut_cdf_qi, 9
cut_cdf_qi(), 9
discrete_scale(), 67
dist_beta(), 17, 76, 84, 92, 101, 109, 117, 124, 129, 135, 140, 146, 155
dist_normal(), 17, 18, 76, 77, 84, 85, 92, 94, 101, 102, 109, 110, 117, 119, 124, 125, 129–131, 135, 136, 140–142, 146, 147, 155, 156
dlkjcorr marginal (lkjcorr marginal), 49
dnorm(), 17, 76, 84, 93, 101, 109, 118, 124, 130, 135, 141, 146, 155
dplyr::filter(), 51
dplyr::group_by(), 58
dplyr::select(), 6
dstudent_t (student_t), 160
element_text(), 46, 47

facet_title_horizontal (theme_ggdist), 161

fda::fbplot(), 7
find_dotplot_binwidth, 11
scale_point_colour_continuous (scales), 62
scale_point_colour_discrete (scales), 62
scale_point_fill_continuous (scales), 62
scale_point_fill_discrete (scales), 62
scale_point_size_continuous (scales), 62
scale_point_size_discrete (scales), 62
scale_point_size_continuous (), 16, 31, 41, 74, 82, 91, 99, 107, 116, 134, 152
scale_point_size_discrete (scales), 62
scale_size_continuous (), 15, 23, 30, 41, 42, 73, 82, 90, 98, 107, 115, 123, 133, 152
scale_slab_alpha_continuous (scales), 62
scale_slab_alpha_discrete (scales), 62
scale_slab_color_continuous (scales), 62
scale_slab_color_discrete (scales), 62
scale_slab_colour_continuous (scales), 62
scale_slab_colour_discrete (scales), 62
scale_slab_fill_continuous (scales), 62
scale_slab_fill_discrete (scales), 62
scale_slab_linetype_continuous (scales), 62
scale_slab_linetype_discrete (scales), 62
scale_slab_shape_continuous (scales), 62
scale_slab_shape_discrete (scales), 62
scale_slab_size_continuous (scales), 62
scale_thickness, 66, 68, 69
scale_thickness_identity (scale_thickness), 69
scale_thickness_identity (), 70
scale_thickness_shared (scale_thickness), 69
scale_thickness_shared (), 70
scales::extended_breaks (), 69
scales::percent_format (), 9
stat (), 76, 84, 93, 101, 110, 118, 124, 130, 135, 141, 147, 155
stat_ccdfinterval (), 71, 87, 96, 104, 113, 121, 126, 137, 149
stat_ccdfinterval (), 70, 73, 154
stat_ccdfinterval, 79, 80, 96, 104, 113, 121, 126, 137, 149
stat_ccdfinterval (), 81, 154
stat_dist_ccdfinterval (ggdist-deprecated), 45
stat_dist_ccdfinterval (ggdist-deprecated), 45
stat_dist_dots (ggdist-deprecated), 45
stat_dist_dotsinterval (ggdist-deprecated), 45
stat_dist_halfeye (ggdist-deprecated), 45
stat_dist_interval (ggdist-deprecated), 45
stat_dist_lineribbon (ggdist-deprecated), 45
stat_dist_pointinterval (ggdist-deprecated), 45
stat_dist_slab (ggdist-deprecated), 45
stat_dist_slabinterval (ggdist-deprecated), 45
stat_dots (geom_dotsinterval), 12
stat_dots (), 12, 17
stat_dotsinterval (geom_dotsinterval), 12
stat_dotsinterval (), 3, 12, 17
stat_eye, 79, 87, 88, 104, 113, 121, 126, 137, 149
stat_eye (), 58, 89, 154
stat_gradientinterval, 79, 87, 96, 97, 113, 121, 126, 137, 149
stat_gradientinterval (), 35, 40, 65, 70, 73, 81, 90, 98, 99, 107, 115, 144, 152, 154
stat_halfeye, 79, 87, 96, 104, 105, 121, 126, 137, 149
stat_halfeye (), 44, 52, 58, 106, 154
stat_histinterval, 79, 87, 96, 104, 113, 113, 126, 137, 149
stat_histinterval (), 114, 154
stat_interval, 79, 87, 96, 104, 113, 121, 122, 137, 149
stat_interval (), 25, 123, 154
stat_lineribbon, 127, 142
stat_lineribbon(), 3, 28, 127, 128
stat_pointinterval, 79, 87, 96, 104, 113,
121, 126, 132, 149
stat_pointinterval(), 33, 133, 154
stat.ribbon, 131, 138
stat.ribbon(), 138, 139
stat_sample_slabinterval
  (ggdist-deprecated), 45
stat_slab, 79, 87, 96, 104, 113, 121, 126,
137, 143
stat_slab(), 38, 144, 154
stat_slabinterval, 150
stat_slabinterval(), 3, 9, 12, 16, 20, 42,
44–46, 50–52, 54, 70, 72, 79, 80, 87,
88, 96, 97, 104, 105, 113, 121, 122,
126, 127, 132, 137, 138, 143, 149,
151, 160
stat_summary(), 58
StatDistSlabinterval
  (ggdist-deprecated), 45
StatSampleSlabinterval
  (ggdist-deprecated), 45
student_t, 160
theme(), 46, 47
theme_ggdist, 161
theme_ggdist(), 162
theme_tidybayes (theme_ggdist), 161
theme_tidybayes(), 162
thickness (scale_thickness), 69
thickness(), 70
tidy-format-translators, 162
to_broom_names
  (tidy-format-translators), 162
to_broom_names(), 163
to_ggmcmc_names
  (tidy-format-translators), 162
to_ggmcmc_names(), 163
transformation object, 69
ul (point_interval), 54
unit, 14
unit(), 14, 15
waiver(), 46