

# Package ‘ggdist’

February 27, 2022

**Title** Visualizations of Distributions and Uncertainty

**Version** 3.1.1

**Date** 2022-02-26

**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:10.1080/00031305.1999.10474474](https://doi.org/10.1080/00031305.1999.10474474)>, quantile dot plots (Kay M., Kola T., Hullman J., Munson S., 2016) <[doi:10.1145/2858036.2858558](https://doi.org/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](https://doi.org/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** 7.1.2

**LazyData** true

**Encoding** UTF-8

**Collate** `ggdist-package.R` `util.R` `abstract\_geom.R`  
`abstract\_stat.R` `abstract\_stat\_slabinterval.R`  
`binning\_methods.R` `curve\_interval.R` `cut\_cdf\_qi.R` `data.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`  
`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\_.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** 2022-02-27 06:00:02 UTC

## R topics documented:

ggdist-package	3
bin_dots	4
curve_interval	6
cut_cdf_qi	9
find_dotplot_binwidth	10
geom_dotsinterval	12
geom_interval	21
geom_lineribbon	25
geom_pointinterval	28
geom_slab	33
geom_slabinterval	38
ggdist-deprecated	44
lkjcorr_marginal	45
marginalize_lkjcorr	47
parse_dist	49
point_interval	51

position_dodgejust . . . . .	56
scales . . . . .	58
scale_colour_ramp . . . . .	63
stat_ccdfinterval . . . . .	65
stat_cdfinterval . . . . .	73
stat_eye . . . . .	82
stat_gradientinterval . . . . .	90
stat_halfeye . . . . .	98
stat_histinterval . . . . .	106
stat_interval . . . . .	115
stat_lineribbon . . . . .	120
stat_pointinterval . . . . .	125
stat_ribbon . . . . .	131
stat_slab . . . . .	135
stat_slabinterval . . . . .	142
student_t . . . . .	152
theme_ggdist . . . . .	153
tidy-format-translators . . . . .	154

<b>Index</b>	<b>156</b>
--------------	------------

---

ggdist-package	<i>Visualizations of Distributions and Uncertainty</i>
----------------	--

---

## 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: <ul style="list-style-type: none"> <li>• "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.</li> <li>• "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".</li> <li>• "swarm": uses the "compactswarm" layout from <code>beeswarm::beeswarm()</code>. 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").</li> </ul>

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 <code>geom_slabinterval()</code> : <ul style="list-style-type: none"> <li>• "horizontal" assumes the data values for the dotplot are in the x variable and that dots will be stacked up in the y direction.</li> <li>• "vertical" assumes the data values for the dotplot are in the y variable and that dots will be stacked up in the x direction.</li> </ul> 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
# 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()
```

---

curve_interval	<i>Curvewise point and interval summaries for tidy data frames of draws from distributions</i>
----------------	--

---

## 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

```
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 <code>group_by()</code> ) that contains draws to summarize.
...	Bare column names or expressions that, when evaluated in the context of <code>.data</code> , represent draws to summarize. If this is empty, then by default all columns that are not group columns and which are not in <code>.exclude</code> (by default <code>".chain"</code> , <code>".iteration"</code> , <code>".draw"</code> , and <code>".row"</code> ) 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 <code>dplyr::select()</code> . Intervals are calculated jointly with respect to these variables, conditional on all other grouping variables in the data frame. The default (NULL) causes <code>curve_interval()</code> to use all grouping variables in the input data frame as the value for <code>.along</code> , 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 <code>g</code> ), you would group by <code>g</code> , then use <code>.along = x</code> to calculate intervals jointly over <code>x</code> conditional on <code>g</code> .
.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 <code>.width</code> column).

<code>.interval</code>	<p>The method used to calculate the intervals. Currently, all methods rank the curves using some measure of <i>data depth</i>, then create envelopes containing the <code>.width%</code> "deepest" curves. Available methods are:</p> <ul style="list-style-type: none"> <li>• "mhd": mean halfspace depth (Fraiman and Muniz 2001).</li> <li>• "mbd": modified band depth (Sun and Genton 2011): calls <code>fda::fbplot()</code> with <code>method = "MBD"</code>.</li> <li>• "bd": band depth (Sun and Genton 2011): calls <code>fda::fbplot()</code> with <code>method = "BD2"</code>.</li> <li>• "bd-mbd": band depth, breaking ties with modified band depth (Sun and Genton 2011): calls <code>fda::fbplot()</code> with <code>method = "Both"</code>.</li> </ul>
<code>.simple_names</code>	<p>When TRUE and only a single column / vector is to be summarized, use the name <code>.lower</code> for the lower end of the interval and <code>.upper</code> for the upper end. If <code>.data</code> is a vector and this is TRUE, this will also set the column name of the point summary to <code>.value</code>. When FALSE and <code>.data</code> is a data frame, names the lower and upper intervals for each column <code>x</code> <code>x.lower</code> and <code>x.upper</code>. When FALSE and <code>.data</code> is a vector, uses the naming scheme <code>y</code>, <code>ymin</code> and <code>ymax</code> (for use with <code>ggplot</code>).</p>
<code>na.rm</code>	<p>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 <code>.interval</code> is "mhd"; other methods do not currently support <code>na.rm</code>. Be cautious in applying this parameter: in general, it is unclear what a joint interval should be when any of the values are missing!</p>
<code>.exclude</code>	<p>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 <code>tidybayes</code>.</p>

## Details

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.

## 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`).

**Author(s)**

Matthew Kay

**References**

Fraiman, Ricardo and Graciela Muniz. (2001). "Trimmed means for functional data". *Test* 10: 419–440. doi: [10.1007/BF02595706](https://doi.org/10.1007/BF02595706).

Sun, Ying and Marc G. Genton. (2011). "Functional Boxplots". *Journal of Computational and Graphical Statistics*, 20(2): 316-334. doi: [10.1198/jcgs.2011.09224](https://doi.org/10.1198/jcgs.2011.09224)

Mirzargar, Mahsa, Ross T Whitaker, and Robert M Kirby. (2014). "Curve Boxplot: Generalization of Boxplot for Ensembles of Curves". *IEEE Transactions on Visualization and Computer Graphics*. 20(12): 2654-2663. doi: [10.1109/TVCG.2014.2346455](https://doi.org/10.1109/TVCG.2014.2346455)

Juul Jonas, Kaare Græsbøll, Lasse Engbo Christiansen, and Sune Lehmann. (2020). "Fixed-time descriptive statistics underestimate extremes of epidemic curve ensembles". *arXiv e-print*. [arXiv:2007.05035](https://arxiv.org/abs/2007.05035)

**See Also**

[point\\_interval\(\)](#) for pointwise intervals. See `vignette("lineribbon")` for more examples and discussion of the differences between pointwise and curvewise intervals.

**Examples**

```
library(dplyr)
library(ggplot2)

# generate a set of curves
k = 11 # number of curves
n = 201
df = tibble(
  .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
```



```
df %>%
  group_by(x) %>%
  curve_interval(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% curvewise intervals with curve_interval()") +
  theme_ggdist()
```

cut\_cdf\_qi

*Categorize 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. Useful for drawing slabs with intervals overlaid on the density, e.g. using [stat\\_halfeye\(\)](#).

**Usage**

```
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. <a href="#">pnorm()</a> ), the <a href="#">cdf()</a> function, or values of the cdf computed aesthetic from the <a href="#">stat_slabininterval()</a> family of stats.
.width	vector of probabilities to use that determine the widths of the resulting intervals.
labels	One of: <ul style="list-style-type: none"> <li>• NULL to use the default labels (.width converted to a character vector).</li> <li>• A character vector giving labels (must be same length as .width)</li> <li>• A function that takes numeric probabilities as input and returns labels as output (a good candidate might be <a href="#">scales::percent_format()</a>).</li> </ul>

**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\\_slabininterval\(\)](#) and its shortcut stats, which generate cdf aesthetics that can be used with [cut\\_cdf\\_qi\(\)](#) to draw slabs colored by their intervals.

**Examples**

```

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

theme_set(theme_ggdist())

# with a slab
tibble(x = dist_normal(0, 1)) %>%
  ggplot(aes(xdist = x)) +
  stat_slab(aes(
    fill = stat(cut_cdf_qi(cdf))
  )) +
  scale_fill_brewer(direction = -1, na.value = "gray90")

# 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 *Dynamically select 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**

```
find_dotplot_binwidth(x, maxheight, heightratio = 1, 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**

```
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	<i>Automatic dotplots, dots + intervals, and quantile dotplots (ggplot geom)</i>
-------------------	--

---

### 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

```
geom_dotsinterval(  
  mapping = NULL,  
  data = NULL,  
  stat = "identity",  
  position = "identity",  
  ...,  
  binwidth = NA,  
  dotsize = 1.07,  
  stackratio = 1,  
  layout = "bin",  
  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  
)
```

```
geom_dots(  
  mapping = NULL,  
  data = NULL,  
  stat = "identity",  
  position = "identity",  
  ...,  
  binwidth = NA,  
  dotsize = 1.07,  
  stackratio = 1,  
  layout = "bin",  
  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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (the default), it is combined with the default mapping at the top level of the plot. You must supply <code>mapping</code> if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
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" ( <code>position_dodge()</code> ) or "dodgejust" ( <code>position_dodgejust()</code> ) can be useful if you have overlapping geometries.
...	Other arguments passed to <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> (see <b>Aesthetics</b> , below). They may also be parameters to the paired geom/stat.
binwidth	<p>The bin width to use for laying out the dots. One of:</p> <ul style="list-style-type: none"> <li>• 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).</li> <li>• A length-1 (scalar) numeric or <code>unit</code> object giving the exact bin width.</li> <li>• A length-2 (vector) numeric or <code>unit</code> object giving the minimum and maximum desired bin width. The bin width will be dynamically selected within these bounds.</li> </ul> <p>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 <code>unit()</code>, 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, <code>unit(0.1, "npc")</code> would make dots that are <i>exactly</i> 10% of the viewport size along whichever dimension the dotplot is drawn; <code>unit(c(0, 0.1), "npc")</code> would make dots that are <i>at most</i> 10% of the viewport size (while still ensuring the tallest stack is less than or equal to scale).</p>
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 <i>precisely</i> the binwidth). If it is desired to have dots be precisely the binwidth, set <code>dotsize = 1</code> .
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.
layout	<p>The layout method used for the dots:</p> <ul style="list-style-type: none"> <li>• "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.</li> <li>• "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 <code>side = "both"</code>.</li> <li>• "swarm": uses the "compactswarm" layout from <code>beeswarm: :beeswarm()</code>. 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").</li> </ul>

orientation	<p>Whether this geom is drawn horizontally or vertically. One of:</p> <ul style="list-style-type: none"> <li>• NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.</li> <li>• "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.</li> <li>• "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.</li> </ul> <p>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).</p>
interval_size_domain	<p>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.)</p>
interval_size_range	<p>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 <code>scale_size_continuous()</code>, which give sizes with a range of <code>c(1,6)</code>. The <code>interval_size_domain</code> value indicates the input domain of raw size values (typically this should be equal to the value of the range argument of the <code>scale_size_continuous()</code> function), and <code>interval_size_range</code> 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 <code>interval_size</code> or <code>point_size</code> aesthetics; see <a href="#">scales</a>.</p>
fatten_point	<p>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 <code>point_size</code> aesthetic and <code>scale_point_size_continuous()</code> or <code>scale_point_size_discrete()</code>; sizes specified with that aesthetic will not be adjusted using <code>fatten_point</code>.</p>
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. <code>borders()</code> .
geom	Use to override the default connection between <code>stat_slabinterval()</code> and <code>geom_slabinterval()</code>
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 <code>point_interval()</code> family (e.g., <code>median_qi</code> , <code>mean_qi</code> , <code>mode_hdi</code> , 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 <code>ggdist</code> 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, hdc). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of orientation. See the <code>point_interval()</code> family of functions for more information.
.width	The <code>.width</code> argument passed to <code>point_interval</code> : 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 <code>.width</code> 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.

### 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.
- `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)

- `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**

Kay, M., Kola, T., Hullman, J. R., & Munson, S. A. (2016). When (ish) is My Bus? User-centered Visualizations of Uncertainty in Everyday, Mobile Predictive Systems. *Conference on Human Factors in Computing Systems - CHI '16*, 5092–5103. doi: [10.1145/2858036.2858558](https://doi.org/10.1145/2858036.2858558).

Fernandes, M., Walls, L., Munson, S., Hullman, J., & Kay, M. (2018). Uncertainty Displays Using Quantile Dotplots or CDFs Improve Transit Decision-Making. *Conference on Human Factors in Computing Systems - CHI '18*. doi: [10.1145/3173574.3173718](https://doi.org/10.1145/3173574.3173718).

**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**

```
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	<i>Multiple-interval plot (shortcut geom)</i>
---------------	---

---

### Description

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

Roughly equivalent to:

```
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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
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" ( <a href="#">position_dodge()</a> ) or "dodgejust" ( <a href="#">position_dodgejust()</a> ) can be useful if you have overlapping geometries.
...	Other arguments passed to <a href="#">layer()</a> . These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3 (see <b>Aesthetics</b> , below). They may also be parameters to the paired geom/stat.
orientation	Whether this geom is drawn horizontally or vertically. One of: <ul style="list-style-type: none"> <li>• NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.</li> <li>• "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.</li> <li>• "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.</li> </ul> <p>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).</p>
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 <a href="#">scale_size_continuous()</a> , 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 <a href="#">scale_size_continuous()</a> 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 <a href="#">scales</a> .
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. <a href="#">borders()</a> .

## 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

```
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()

```

---

geom\_lineribbon

*Line + multiple-ribbon plots (ggplot geom)*


---

## Description

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

## Usage

```

geom_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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> .

	<p>A <code>data.frame</code>, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p>
<code>stat</code>	The statistical transformation to use on the data for this layer, as a string.
<code>position</code>	Position adjustment, either as a string, or the result of a call to a position adjustment function.
<code>...</code>	Other arguments passed to <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> (see <b>Aesthetics</b> , below). They may also be parameters to the paired <code>geom/stat</code> .
<code>step</code>	<p>Should the line/ribbon be drawn as a step function? One of:</p> <ul style="list-style-type: none"> <li>• <code>FALSE</code> (default): do not draw as a step function.</li> <li>• <code>"mid"</code> (or <code>TRUE</code>): draw steps midway between adjacent x values.</li> <li>• <code>"hv"</code>: draw horizontal-then-vertical steps.</li> <li>• <code>"vh"</code>: draw as vertical-then-horizontal steps.</li> </ul> <p><code>TRUE</code> is an alias for <code>"mid"</code> because for a step function with ribbons, <code>"mid"</code> 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).</p>
<code>orientation</code>	<p>Whether this geom is drawn horizontally or vertically. One of:</p> <ul style="list-style-type: none"> <li>• <code>NA</code> (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.</li> <li>• <code>"horizontal"</code> (or <code>"y"</code>): draw horizontally, using the y aesthetic to identify different groups. For each group, uses the x, <code>xmin</code>, <code>xmax</code>, and thickness aesthetics to draw points, intervals, and slabs.</li> <li>• <code>"vertical"</code> (or <code>"x"</code>): draw vertically, using the x aesthetic to identify different groups. For each group, uses the y, <code>ymin</code>, <code>ymax</code>, and thickness aesthetics to draw points, intervals, and slabs.</li> </ul> <p>For compatibility with the base <code>ggplot</code> naming scheme for orientation, <code>"x"</code> can be used as an alias for <code>"vertical"</code> and <code>"y"</code> as an alias for <code>"horizontal"</code> (<code>tidybayes</code> had an <code>orientation</code> parameter before base <code>ggplot</code> did, hence the discrepancy).</p>
<code>na.rm</code>	If <code>FALSE</code> , the default, missing values are removed with a warning. If <code>TRUE</code> , missing values are silently removed.
<code>show.legend</code>	logical. Should this layer be included in the legends? <code>NA</code> , the default, includes if any aesthetics are mapped. <code>FALSE</code> never includes, and <code>TRUE</code> always includes. It can also be a named logical vector to finely select the aesthetics to display.
<code>inherit.aes</code>	If <code>FALSE</code> , 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. <code>borders()</code> .

## Details

`geom_lineribbon()` is a combination of a `geom_line()` and `geom_ribbon()` designed for use with output from `point_interval()`. This geom sets some default aesthetics equal to the `.width` column generated by the `point_interval()` family of functions, making them often more convenient than a vanilla `geom_ribbon() + geom_line()`.

Specifically, `geom_lineribbon()` acts as if its default aesthetics are `aes(fill = forcats::fct_rev(ordered(.width)))`.

## Value

A `ggplot2::Geom` representing a combined line + multiple-ribbon geometry which can be added to a `ggplot()` object.

## 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**.

### Positional aesthetics

- `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**

```
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:

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

**Usage**

```
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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
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" ( <code>position_dodge()</code> ) or "dodgejust" ( <code>position_dodgejust()</code> ) can be useful if you have overlapping geometries.
...	Other arguments passed to <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> (see <b>Aesthetics</b> , below). They may also be parameters to the paired geom/stat.
orientation	Whether this geom is drawn horizontally or vertically. One of: <ul style="list-style-type: none"> <li>• <code>NA</code> (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.</li> <li>• <code>"horizontal"</code> (or <code>"y"</code>): draw horizontally, using the <code>y</code> aesthetic to identify different groups. For each group, uses the <code>x</code>, <code>xmin</code>, <code>xmax</code>, and <code>thickness</code> aesthetics to draw points, intervals, and slabs.</li> <li>• <code>"vertical"</code> (or <code>"x"</code>): draw vertically, using the <code>x</code> aesthetic to identify different groups. For each group, uses the <code>y</code>, <code>ymin</code>, <code>ymax</code>, and <code>thickness</code> aesthetics to draw points, intervals, and slabs.</li> </ul>

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 <code>scale_size_continuous()</code> , which give sizes with a range of <code>c(1,6)</code> . The <code>interval_size_domain</code> value indicates the input domain of raw size values (typically this should be equal to the value of the range argument of the <code>scale_size_continuous()</code> function), and <code>interval_size_range</code> 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 <code>interval_size</code> or <code>point_size</code> aesthetics; see <a href="#">scales</a> .
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 <code>point_size</code> aesthetic and <code>scale_point_size_continuous()</code> or <code>scale_point_size_discrete()</code> ; sizes specified with that aesthetic will not be adjusted using <code>fatten_point</code> .
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 <code>c(size = FALSE)</code> , 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. <code>borders()</code> .

## 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 are `aes(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.
- `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()
```

---

geom_slab	<i>Slab (ridge) plot (shortcut geom)</i>
-----------	--

---

## Description

Shortcut version of `geom_slabinterval()` for creating slab (ridge) plots.

Roughly equivalent to:

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

## Usage

```
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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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:<br>If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> .<br>A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created.<br>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ). |

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" ( <code>position_dodge()</code> ) or "dodgejust" ( <code>position_dodgejust()</code> ) can be useful if you have overlapping geometries.
...	Other arguments passed to <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> (see <b>Aesthetics</b> , below). They may also be parameters to the paired geom/stat.
orientation	<p>Whether this geom is drawn horizontally or vertically. One of:</p> <ul style="list-style-type: none"> <li>• NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.</li> <li>• "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.</li> <li>• "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.</li> </ul> <p>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).</p>
normalize	<p>How to normalize heights of functions input to the thickness aesthetic. One of:</p> <ul style="list-style-type: none"> <li>• "all": normalize so that the maximum height across all data is 1.</li> <li>• "panels": normalize within panels so that the maximum height in each panel is 1.</li> <li>• "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.</li> <li>• "groups": normalize within values of the opposite axis and within each group so that the maximum height in each group is 1.</li> <li>• "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).</li> </ul>
fill_type	<p>What type of fill to use when the fill color or alpha varies within a slab. One of:</p> <ul style="list-style-type: none"> <li>• "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 <code>stat_gradientinterval()</code>).</li> <li>• "gradient": a <code>grid::linearGradient()</code> is used to create a smooth gradient fill. This works well for large numbers of unique fill colors, but requires R <math>\geq</math> 4.1 and is not yet supported on all graphics devices. As of this writing, the <code>png()</code> graphics device with <code>type = "cairo"</code>, the <code>svg()</code> device, the <code>pdf()</code> device, and the <code>ragg::agg_png()</code> devices are known to support this option. On R &lt; 4.1, this option will fall back to <code>fill_type = "segment"</code> with a message.</li> </ul>

	<ul style="list-style-type: none"> <li>• "auto": attempts to use <code>fill_type = "gradient"</code> if support for it can be auto-detected. On R <math>\geq</math> 4.2, support for gradients can be auto-detected on some graphics devices; if support is not detected, this option will fall back to <code>fill_type = "segments"</code> (in case of a false negative, <code>fill_type = "gradient"</code> can be set explicitly). On R <math>&lt;</math> 4.2, support for gradients cannot be auto-detected, so this will always fall back to <code>fill_type = "segments"</code>, in which case you can set <code>fill_type = "gradient"</code> explicitly if you are using a graphics device that support gradients.</li> </ul>
<code>na.rm</code>	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
<code>show.legend</code>	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.
<code>inherit.aes</code>	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. <code>borders()</code> .

## 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.

### 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

```
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

```
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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (the default), it is combined with the default mapping at the top level of the plot. You must supply <code>mapping</code> if there is no plot mapping.
data	<p>The data to be displayed in this layer. There are three options:</p> <p>If <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code>.</p> <p>A <code>data.frame</code>, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p>

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" ( <a href="#">position_dodge()</a> ) or "dodgejust" ( <a href="#">position_dodgejust()</a> ) can be useful if you have overlapping geometries.
...	Other arguments passed to <a href="#">layer()</a> . These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3 (see <b>Aesthetics</b> , below). They may also be parameters to the paired geom/stat.
orientation	<p>Whether this geom is drawn horizontally or vertically. One of:</p> <ul style="list-style-type: none"> <li>• NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.</li> <li>• "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.</li> <li>• "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.</li> </ul> <p>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).</p>
normalize	<p>How to normalize heights of functions input to the thickness aesthetic. One of:</p> <ul style="list-style-type: none"> <li>• "all": normalize so that the maximum height across all data is 1.</li> <li>• "panels": normalize within panels so that the maximum height in each panel is 1.</li> <li>• "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.</li> <li>• "groups": normalize within values of the opposite axis and within each group so that the maximum height in each group is 1.</li> <li>• "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).</li> </ul>
fill_type	<p>What type of fill to use when the fill color or alpha varies within a slab. One of:</p> <ul style="list-style-type: none"> <li>• "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 <a href="#">stat_gradientinterval()</a>).</li> <li>• "gradient": a <code>grid::linearGradient()</code> is used to create a smooth gradient fill. This works well for large numbers of unique fill colors, but requires R <math>\geq</math> 4.1 and is not yet supported on all graphics devices. As of this writing, the <code>png()</code> graphics device with <code>type = "cairo"</code>, the <code>svg()</code> device, the <code>pdf()</code> device, and the <code>ragg::agg_png()</code> devices are known to support this option. On R &lt; 4.1, this option will fall back to <code>fill_type = "segment"</code> with a message.</li> </ul>

- "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 <code>scale_size_continuous()</code> , which give sizes with a range of <code>c(1,6)</code> . The <code>interval_size_domain</code> value indicates the input domain of raw size values (typically this should be equal to the value of the range argument of the <code>scale_size_continuous()</code> function), and <code>interval_size_range</code> 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 <code>interval_size</code> or <code>point_size</code> aesthetics; see <a href="#">scales</a> .
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 <code>point_size</code> aesthetic and <code>scale_point_size_continuous()</code> or <code>scale_point_size_discrete()</code> ; sizes specified with that aesthetic will not be adjusted using <code>fatten_point</code> .
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. <code>borders()</code> .

## 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.
```

---

ggdist-deprecated      *Deprecated functions and arguments in ggdist*

---

**Description**

Deprecated 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 = \((...) point_interval_arg = XXX)`

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

---

lkjcorr_marginal	<i>Marginal distribution of a single correlation from an LKJ distribution</i>
------------------	---

---

### 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 = FALSE)
```

```
plkjcorr_marginal(q, K, eta, lower.tail = TRUE, log.p = FALSE)
```

```
qlkjcorr_marginal(p, K, eta, lower.tail = TRUE, log.p = FALSE)
```

```
rlkjcorr_marginal(n, K, eta)
```

**Arguments**

x	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	logical; if TRUE, probabilities p are given as log(p).
q	vector of quantiles.
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$ otherwise, $P[X > x]$ .
log.p	logical; if TRUE, probabilities p are given as log(p).
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**

- `dlkcorr_marginal` gives the density
- `plkcorr_marginal` gives the cumulative distribution function (CDF)
- `qlkcorr_marginal` gives the quantile function (inverse CDF)
- `rlkcorr_marginal` generates random draws.

The length of the result is determined by `n` for `rlkcorr_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**

Lewandowski, D., Kurowicka, D., & Joe, H. (2009). Generating random correlation matrices based on vines and extended onion method. *Journal of Multivariate Analysis*, 100(9), 1989–2001. doi: [10.1016/j.jmva.2009.04.008](https://doi.org/10.1016/j.jmva.2009.04.008).

**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**

```
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**

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

**Arguments**

data	A data frame containing a column with distribution names (" <code>.dist</code> " by default) and a list column of distribution arguments (" <code>.args</code> " by default), such as output by <code>parse_dist()</code> .
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 <code>dplyr::filter()</code> ) that evaluates to TRUE on the rows you want to modify. If NULL (the default), all <code>lkjcorr</code> distributions in data are modified.
dist	The name of the column containing distribution names. See <code>parse_dist()</code> .
args	The name of the column containing distribution arguments. See <code>parse_dist()</code> .

**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)

# Say we have an LKJ(3) prior on a 2x2 correlation matrix. We can visualize
# its marginal distribution as follows...
data.frame(prior = "lkjcorr(3)") %>%
  parse_dist(prior) %>%
  marginalize_lkcorr(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")

# Say our prior list has multiple LKJ priors on correlation matrices
# of different sizes, we can supply a predicate expression to select
# only those rows we want to modify
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,
  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 <code>parse_dist</code> .
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 <code>r_dist_name</code> 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 <code>r_dist_name</code> , 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**

```
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	<i>Point and interval summaries for tidy data frames of draws from distributions</i>
----------------	--

---

**Description**

Translates draws from distributions in a (possibly grouped) data frame into point and interval summaries (or set of point and interval summaries, if there are multiple groups in a grouped data frame).

**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  
)
```

```
## 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)
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**

<code>.data</code>	Data frame (or grouped data frame as returned by <code>group_by()</code> ) that contains draws to summarize.
<code>...</code>	Bare column names or expressions that, when evaluated in the context of <code>.data</code> , represent draws to summarize. If this is empty, then by default all columns that are not group columns and which are not in <code>.exclude</code> (by default <code>".chain"</code> , <code>".iteration"</code> , <code>".draw"</code> , and <code>".row"</code> ) will be summarized. These columns can be numeric, <b>distributional</b> objects, <code>posterior::rvars</code> , or list columns of numeric values to summarise.
<code>.width</code>	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 <code>.width</code> column).
<code>.point</code>	Point summary function, which takes a vector and returns a single value, e.g. <code>mean()</code> , <code>median()</code> , or <code>Mode()</code> .
<code>.interval</code>	Interval function, which takes a vector and a probability ( <code>.width</code> ) and returns a two-element vector representing the lower and upper bound of an interval; e.g. <code>qi()</code> , <code>hdi()</code>
<code>.simple_names</code>	When TRUE and only a single column / vector is to be summarized, use the name <code>.lower</code> for the lower end of the interval and <code>.upper</code> for the upper end. If <code>.data</code> is a vector and this is TRUE, this will also set the column name of the point summary to <code>.value</code> . When FALSE and <code>.data</code> is a data frame, names the lower and upper intervals for each column <code>x.lower</code> and <code>x.upper</code> . When FALSE and <code>.data</code> is a vector, uses the naming scheme <code>y</code> , <code>ymin</code> and <code>ymax</code> (for use with <code>ggplot</code> ).
<code>na.rm</code>	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.
<code>.exclude</code>	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.
<code>.prob</code>	Deprecated. Use <code>.width</code> instead.
<code>x</code>	vector to summarize (for interval functions: <code>qi</code> and <code>hdi</code> )

**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 `HDInterval::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 `HDInterval::hdi()` with `allowSplit = FALSE`; see that function for more information on multimodality and continuous versus discontinuous intervals.

`l1` and `u1` 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`).

## 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

```

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**

```

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.
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_dodge(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.
dist_df %>%
  ggplot(aes(
    x = factor(group), ydist = dist_normal(mean, sd),
    fill = subgroup
  )) +
  stat_halfeye(
    position = "dodgejust"
  ) +
  geom_rect(
    aes(xmin = group, xmax = group + 1, ymin = 2, ymax = 13, color = subgroup),
    position = "dodgejust",
    data = . %>% filter(group == 3),
    alpha = 0.1
  ) +
  geom_point(
    aes(x = group, y = 7.5, color = subgroup),
    position = position_dodgejust(width = 1, justification = 0),
    data = . %>% filter(group == 3),
    shape = 1,
    size = 4,
    stroke = 1.5
  ) +
  scale_fill_brewer(palette = "Set2") +
  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

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

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

scale_point_colour_continuous(
  ...,
  aesthetics = "point_colour",
  guide = "colourbar2"
)

scale_point_color_continuous(
  ...,
  aesthetics = "point_colour",
  guide = "colourbar2"
)

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

scale_point_fill_continuous(
  ...,
  aesthetics = "point_fill",
  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 = "colourbar2"
)

scale_interval_color_continuous(
  ...,
  aesthetics = "interval_colour",
  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 = "colourbar2"
)
scale_slab_color_continuous(
  ...,
  aesthetics = "slab_colour",
  guide = "colourbar2"
)
scale_slab_fill_discrete(..., aesthetics = "slab_fill")
scale_slab_fill_continuous(..., aesthetics = "slab_fill", 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. <code>scale_point_color_discrete</code> passes arguments to <code>scale_color_discrete()</code> . 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 <code>na.translate</code> is true, what value should be shown?
limits	One of: <ul style="list-style-type: none"> <li>• NULL to use the default scale range</li> <li>• A numeric vector of length two providing limits of the scale. Use NA to refer to the existing minimum or maximum</li> <li>• A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang <code>lambda</code> function notation. Note that setting limits on positional scales will <b>remove</b> data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see <code>coord_cartesian()</code>).</li> </ul>
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

### See Also

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

Other `ggdist` scales: `scale_colour_ramp`

### Examples

```
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
  )

```

---

scale\_colour\_ramp      *Secondary ggplot color scale that ramps from another color*

---

### 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).

### Usage

```

scale_colour_ramp_continuous(
  from = "white",
  ...,
  limits = function(l) c(min(0, l[[1]]), l[[2]]),
  range = c(0, 1),
  aesthetics = "colour_ramp"
)

```

```

scale_color_ramp_continuous(
  from = "white",
  ...,
  limits = function(l) c(min(0, l[[1]]), l[[2]]),
  range = c(0, 1),
  aesthetics = "colour_ramp"
)

```

```

scale_colour_ramp_discrete(
  from = "white",
  ...,
  range = c(0.2, 1),
  aesthetics = "colour_ramp"
)

```

```

scale_color_ramp_discrete(
  from = "white",

```

```

    ...,
    range = c(0.2, 1),
    aesthetics = "colour_ramp"
  )

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. <code>scale_colour_ramp_discrete()</code> , passes arguments to <code>discrete_scale()</code> , <code>scale_colour_ramp_continuous()</code> passes arguments to <code>continuous_scale()</code> . See those functions for more details.
limits	One of: <ul style="list-style-type: none"> <li>• NULL to use the default scale range</li> <li>• A numeric vector of length two providing limits of the scale. Use NA to refer to the existing minimum or maximum</li> <li>• A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang <code>lambda</code> function notation. Note that setting limits on positional scales will <b>remove</b> data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see <code>coord_cartesian()</code>).</li> </ul>
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).
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

Other `ggdist` scales: [scales](#)

### Examples

```

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))

```

---

stat_ccdfinterval	<i>CCDF bar plot (shortcut stat)</i>
-------------------	--------------------------------------

---

### Description

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

Roughly equivalent to:

```

stat_slabinterval(
  aes(justification = stat(0.5), side = stat("topleft")),
  slab_type = "ccdf", normalize = "none", expand = TRUE
)

```

### Usage

```

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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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	<p>The data to be displayed in this layer. There are three options:</p> <p>If <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code>.</p> <p>A <code>data.frame</code>, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p>
geom	Use to override the default connection between <code>stat_ccdfinterval()</code> and <code>geom_slabinterval()</code>
position	Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" ( <code>position_dodge()</code> ) or "dodgejust" ( <code>position_dodgejust()</code> ) can be useful if you have overlapping geometries.
...	Other arguments passed to <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> (see <b>Aesthetics</b> , below). They may also be parameters to the paired geom/stat. When paired with the default geom, <code>geom_slabinterval()</code> , these include:
fill_type	<p>What type of fill to use when the fill color or alpha varies within a slab. One of:</p> <ul style="list-style-type: none"> <li>"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 <code>stat_gradientinterval()</code>).</li> <li>"gradient": a <code>grid::linearGradient()</code> is used to create a smooth gradient fill. This works well for large numbers of unique fill colors, but requires R <math>\geq</math> 4.1 and is not yet supported on all graphics devices. As of this writing, the <code>png()</code> graphics device with <code>type = "cairo"</code>, the <code>svg()</code> device, the <code>pdf()</code> device, and the <code>ragg::agg_png()</code> devices are known to support this option. On R <math>&lt;</math> 4.1, this option will fall back to <code>fill_type = "segment"</code> with a message.</li> </ul>

- "auto": attempts to use `fill_type = "gradient"` if support for it can be auto-detected. On R  $\geq$  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.

<code>p_limits</code>	Probability limits (as a vector of size 2) used to determine the lower and upper limits of the slab. E.g., if this is <code>c(.001, .999)</code> , 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 <code>NA</code> , then the lower (upper) limit will be the minimum (maximum) of the distribution's support if it is finite, and <code>0.001</code> ( <code>0.999</code> ) if it is not finite. E.g., if <code>p_limits</code> is <code>c(NA, NA)</code> on a gamma distribution the effective value of <code>p_limits</code> would be <code>c(0, .999)</code> since the gamma distribution is defined on $(0, \text{Inf})$ ; whereas on a normal distribution it would be equivalent to <code>c(.001, .999)</code> since the normal distribution is defined on $(-\text{Inf}, \text{Inf})$ .
<code>adjust</code>	If <code>slab_type</code> is "pdf", bandwidth for the density estimator for sample data is adjusted by multiplying it by this value. See <code>density()</code> for more information.
<code>trim</code>	For sample data, should the density estimate be trimmed to the range of the input data? Default TRUE.
<code>breaks</code>	If <code>slab_type</code> is "histogram", the <code>breaks</code> parameter that is passed to <code>hist()</code> to determine where to put breaks in the histogram (for sample data).
<code>outline_bars</code>	For sample data (if <code>slab_type</code> is "histogram") and for discrete analytical distributions (whose slabs are drawn as histograms), determines if outlines in between the bars are drawn when the <code>slab_color</code> 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.
<code>point_interval</code>	A function from the <code>point_interval()</code> family (e.g., <code>median_qi</code> , <code>mean_qi</code> , <code>mode_hdi</code> , 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 <code>ggdist</code> 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, hdc). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of <code>orientation</code> . See the <code>point_interval()</code> family of functions for more information.
<code>limits</code>	Manually-specified limits for the slab, as a vector of length two. These limits are combined with those computed based on <code>p_limits</code> 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 <code>NA</code> to leave a limit alone; e.g. <code>limits = c(0, NA)</code> will ensure that the lower limit does not go below 0, but let the upper limit be determined by either <code>p_limits</code> or the scale settings.
<code>n</code>	Number of points at which to evaluate the function that defines the slab.
<code>.width</code>	The <code>.width</code> argument passed to <code>point_interval</code> : 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 <code>.width</code> and <code>level</code> generated variables).
<code>orientation</code>	Whether this geom is drawn horizontally or vertically. One of: <ul style="list-style-type: none"> <li>• <code>NA</code> (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.</li> </ul>

- "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 <code>c(size = FALSE)</code> , 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. <code>borders()</code> .

## 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  $[0, 1]$ .
- `level`: For intervals, the interval width as an ordered factor.
- `f`: For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by `slab_type`.
- `pdf`: For slabs, the probability density function.
- `cdf`: For slabs, the cumulative distribution function.
- `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\\_cdfinterval\(\)](#), [stat\\_eye\(\)](#), [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_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_ccdfinterval() +
  expand_limits(x = 0)
```

---

<code>stat_cdfinterval</code>	<i>CDF bar plot (shortcut stat)</i>
-------------------------------	-------------------------------------

---

**Description**

Shortcut version of [stat\\_slabinterval\(\)](#) with [geom\\_slabinterval\(\)](#) for creating CDF bar plots.

Roughly equivalent to:

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

**Usage**

```

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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
geom	Use to override the default connection between <code>stat_cdfinterval()</code> and <code>geom_slabinterval()</code>
position	Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" ( <code>position_dodge()</code> ) or "dodgejust" ( <code>position_dodgejust()</code> ) can be useful if you have overlapping geometries.
...	Other arguments passed to <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> (see <b>Aesthetics</b> , below). They may also be parameters to the paired geom/stat. When paired with the default geom, <code>geom_slabinterval()</code> , 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	<p>How to normalize heights of functions input to the thickness aesthetic. One of:</p> <ul style="list-style-type: none"> <li>• "all": normalize so that the maximum height across all data is 1.</li> <li>• "panels": normalize within panels so that the maximum height in each panel is 1.</li> <li>• "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.</li> <li>• "groups": normalize within values of the opposite axis and within each group so that the maximum height in each group is 1.</li> <li>• "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).</li> </ul>
expand	For sample data, should the slab be expanded to the limits of the scale? Default FALSE.
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 <code>c(.001, .999)</code> , 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 <code>p_limits</code> is <code>c(NA, NA)</code> on a gamma distribution the effective value of <code>p_limits</code> would be <code>c(0, .999)</code> since the gamma distribution is defined on $(0, \text{Inf})$ ; whereas on a normal distribution it would be equivalent to <code>c(.001, .999)</code> since the normal distribution is defined on $(-\text{Inf}, \text{Inf})$ .
adjust	If <code>slab_type</code> is "pdf", bandwidth for the density estimator for sample data is adjusted by multiplying it by this value. See <a href="#">density()</a> for more information.
trim	For sample data, should the density estimate be trimmed to the range of the input data? Default TRUE.
breaks	If <code>slab_type</code> is "histogram", the breaks parameter that is passed to <a href="#">hist()</a> to determine where to put breaks in the histogram (for sample data).
outline_bars	For sample data (if <code>slab_type</code> is "histogram") and for discrete analytical distributions (whose slabs are drawn as histograms), determines if outlines in between the bars are drawn when the <code>slab_color</code> 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 <a href="#">point_interval()</a> family (e.g., <code>median_qi</code> , <code>mean_qi</code> , <code>mode_hdi</code> , 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 <code>ggdist</code> 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, hdc). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of orientation. See the <a href="#">point_interval()</a> 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 <code>p_limits</code> 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. <code>limits = c(0, NA)</code> will ensure that the lower limit does not go below 0, but let the upper limit be determined by either <code>p_limits</code> or the scale settings.
<code>n</code>	Number of points at which to evaluate the function that defines the slab.
<code>.width</code>	The <code>.width</code> argument passed to <code>point_interval</code> : 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 <code>.width</code> and level generated variables).
<code>orientation</code>	Whether this geom is drawn horizontally or vertically. One of: <ul style="list-style-type: none"> <li>• NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.</li> <li>• "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.</li> <li>• "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.</li> </ul> <p>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).</p>
<code>na.rm</code>	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
<code>show.legend</code>	Should this layer be included in the legends? Default is <code>c(size = FALSE)</code> , 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).
<code>inherit.aes</code>	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. <code>borders()</code> .

## 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 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  $[0, 1]$ .
- `level`: For intervals, the interval width as an ordered factor.
- `f`: For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by `slab_type`.
- `pdf`: For slabs, the probability density function.
- `cdf`: For slabs, the cumulative distribution function.
- `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\\_eye\(\)](#), [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_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	<i>Eye (violin + interval) plot (shortcut stat)</i>
----------	---

---

### Description

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

Roughly equivalent to:

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

### Usage

```
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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> .

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  $\geq$  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  $\geq$  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`.

<code>slab_type</code>	The type of slab function to calculate: probability density (or mass) function ("pdf"), cumulative distribution function ("cdf"), or complementary CDF ("ccdf").
<code>p_limits</code>	Probability limits (as a vector of size 2) used to determine the lower and upper limits of the slab. E.g., if this is <code>c(.001, .999)</code> , 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 <code>0.001</code> ( <code>0.999</code> ) if it is not finite. E.g., if <code>p_limits</code> is <code>c(NA, NA)</code> on a gamma distribution the effective value of <code>p_limits</code> would be <code>c(0, .999)</code> since the gamma distribution is defined on $(0, \text{Inf})$ ; whereas on a normal distribution it would be equivalent to <code>c(.001, .999)</code> since the normal distribution is defined on $(-\text{Inf}, \text{Inf})$ .
<code>adjust</code>	If <code>slab_type</code> is "pdf", bandwidth for the density estimator for sample data is adjusted by multiplying it by this value. See <a href="#">density()</a> for more information.
<code>trim</code>	For sample data, should the density estimate be trimmed to the range of the input data? Default TRUE.
<code>expand</code>	For sample data, should the slab be expanded to the limits of the scale? Default FALSE.
<code>breaks</code>	If <code>slab_type</code> is "histogram", the breaks parameter that is passed to <a href="#">hist()</a> to determine where to put breaks in the histogram (for sample data).

outline_bars	For sample data (if <code>slab_type</code> is "histogram") and for discrete analytical distributions (whose slabs are drawn as histograms), determines if outlines in between the bars are drawn when the <code>slab_color</code> 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 <code>point_interval()</code> family (e.g., <code>median_qi</code> , <code>mean_qi</code> , <code>mode_hdi</code> , 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 <code>ggdist</code> 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, hdc). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of <code>orientation</code> . See the <code>point_interval()</code> 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 <code>p_limits</code> 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. <code>limits = c(0, NA)</code> will ensure that the lower limit does not go below 0, but let the upper limit be determined by either <code>p_limits</code> or the scale settings.
n	Number of points at which to evaluate the function that defines the slab.
.width	The <code>.width</code> argument passed to <code>point_interval</code> : 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 <code>.width</code> and level generated variables).
orientation	Whether this geom is drawn horizontally or vertically. One of: <ul style="list-style-type: none"> <li>• NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.</li> <li>• "horizontal" (or "y"): draw horizontally, using the <code>y</code> aesthetic to identify different groups. For each group, uses the <code>x</code>, <code>xmin</code>, <code>xmax</code>, and <code>thickness</code> aesthetics to draw points, intervals, and slabs.</li> <li>• "vertical" (or "x"): draw vertically, using the <code>x</code> aesthetic to identify different groups. For each group, uses the <code>y</code>, <code>ymin</code>, <code>ymax</code>, and <code>thickness</code> aesthetics to draw points, intervals, and slabs.</li> </ul> <p>For compatibility with the base <code>ggplot</code> naming scheme for orientation, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal" (<code>tidybayes</code> had an <code>orientation</code> parameter before base <code>ggplot</code> did, hence the discrepancy).</p>
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 <code>c(size = FALSE)</code> , 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 eye (violin + 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].
- `level`: For intervals, the interval width as an ordered factor.
- `f`: For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by `slab_type`.
- `pdf`: For slabs, the probability density function.
- `cdf`: For slabs, the cumulative distribution function.
- `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_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:

```

stat_slabinterval(
  aes(justification = stat(0.5), thickness = stat(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  $\geq$  4.2, support for `fill_type = "gradient"` should be auto-detected based on the graphics device you are using.

## Usage

```

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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
geom	Use to override the default connection between <code>stat_gradientinterval()</code> and <code>geom_slabinterval()</code>
position	Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" ( <code>position_dodge()</code> ) or "dodgejust" ( <code>position_dodgejust()</code> ) can be useful if you have overlapping geometries.
...	Other arguments passed to <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> (see <b>Aesthetics</b> , below). They may also be parameters to the paired geom/stat. When paired with the default geom, <code>geom_slabinterval()</code> , these include:  normalize How to normalize heights of functions input to the thickness aesthetic. One of: <ul style="list-style-type: none"> <li>"all": normalize so that the maximum height across all data is 1.</li> <li>"panels": normalize within panels so that the maximum height in each panel is 1.</li> <li>"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.</li> <li>"groups": normalize within values of the opposite axis and within each group so that the maximum height in each group is 1.</li> <li>"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).</li> </ul> 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 <code>interval_size_range</code> (see the documentation for that argument.)

	<p><code>interval_size_range</code> 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 <code>scale_size_continuous()</code>, which give sizes with a range of <code>c(1,6)</code>. The <code>interval_size_domain</code> value indicates the input domain of raw size values (typically this should be equal to the value of the range argument of the <code>scale_size_continuous()</code> function), and <code>interval_size_range</code> 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 <code>interval_size</code> or <code>point_size</code> aesthetics; see <a href="#">scales</a>.</p>
	<p><code>fatten_point</code> 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 <code>point_size</code> aesthetic and <code>scale_point_size_continuous()</code> or <code>scale_point_size_discrete()</code>; sizes specified with that aesthetic will not be adjusted using <code>fatten_point</code>.</p>
<code>fill_type</code>	<p>What type of fill to use when the fill color or alpha varies within a slab. One of:</p> <ul style="list-style-type: none"> <li>• <code>"segments"</code>: 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 <code>stat_gradientinterval()</code>).</li> <li>• <code>"gradient"</code>: a <code>grid::linearGradient()</code> is used to create a smooth gradient fill. This works well for large numbers of unique fill colors, but requires <code>R &gt;= 4.1</code> and is not yet supported on all graphics devices. As of this writing, the <code>png()</code> graphics device with <code>type = "cairo"</code>, the <code>svg()</code> device, the <code>pdf()</code> device, and the <code>ragg::agg_png()</code> devices are known to support this option. On <code>R &lt; 4.1</code>, this option will fall back to <code>fill_type = "segment"</code> with a message.</li> <li>• <code>"auto"</code>: attempts to use <code>fill_type = "gradient"</code> if support for it can be auto-detected. On <code>R &gt;= 4.2</code>, support for gradients can be auto-detected on some graphics devices; if support is not detected, this option will fall back to <code>fill_type = "segments"</code> (in case of a false negative, <code>fill_type = "gradient"</code> can be set explicitly). On <code>R &lt; 4.2</code>, support for gradients cannot be auto-detected, so this will always fall back to <code>fill_type = "segments"</code>, in which case you can set <code>fill_type = "gradient"</code> explicitly if you are using a graphics device that support gradients.</li> </ul>
<code>slab_type</code>	<p>The type of slab function to calculate: probability density (or mass) function (<code>"pdf"</code>), cumulative distribution function (<code>"cdf"</code>), or complementary CDF (<code>"ccdf"</code>).</p>
<code>p_limits</code>	<p>Probability limits (as a vector of size 2) used to determine the lower and upper limits of the slab. E.g., if this is <code>c(.001, .999)</code>, then a slab is drawn for the distribution from the quantile at <code>p = .001</code> to the quantile at <code>p = .999</code>. If the lower (respectively upper) limit is <code>NA</code>, then the lower (upper) limit will be the minimum (maximum) of the distribution's support if it is finite, and <code>0.001</code> (<code>0.999</code>)</p>

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  $(0, \text{Inf})$ ; whereas on a normal distribution it would be equivalent to `c(.001, .999)` since the normal distribution is defined on  $(-\text{Inf}, \text{Inf})$ .

<code>adjust</code>	If <code>slab_type</code> is "pdf", bandwidth for the density estimator for sample data is adjusted by multiplying it by this value. See <a href="#">density()</a> for more information.
<code>trim</code>	For sample data, should the density estimate be trimmed to the range of the input data? Default TRUE.
<code>expand</code>	For sample data, should the slab be expanded to the limits of the scale? Default FALSE.
<code>breaks</code>	If <code>slab_type</code> is "histogram", the <code>breaks</code> parameter that is passed to <a href="#">hist()</a> to determine where to put breaks in the histogram (for sample data).
<code>outline_bars</code>	For sample data (if <code>slab_type</code> is "histogram") and for discrete analytical distributions (whose slabs are drawn as histograms), determines if outlines in between the bars are drawn when the <code>slab_color</code> 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.
<code>point_interval</code>	A function from the <a href="#">point_interval()</a> family (e.g., <code>median_qi</code> , <code>mean_qi</code> , <code>mode_hdi</code> , 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 <code>ggdist</code> environment). This function determines the point summary (typically mean, median, or mode) and interval type (quantile interval, <code>qi</code> ; highest-density interval, <code>hdi</code> ; or highest-density continuous interval, <code>hdc</code> ). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of <code>orientation</code> . See the <a href="#">point_interval()</a> family of functions for more information.
<code>limits</code>	Manually-specified limits for the slab, as a vector of length two. These limits are combined with those computed based on <code>p_limits</code> 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. <code>limits = c(0,NA)</code> will ensure that the lower limit does not go below 0, but let the upper limit be determined by either <code>p_limits</code> or the scale settings.
<code>n</code>	Number of points at which to evaluate the function that defines the slab.
<code>.width</code>	The <code>.width</code> argument passed to <code>point_interval</code> : 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 <code>.width</code> and <code>level</code> generated variables).
<code>orientation</code>	Whether this geom is drawn horizontally or vertically. One of: <ul style="list-style-type: none"> <li>• NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.</li> <li>• "horizontal" (or "y"): draw horizontally, using the <code>y</code> aesthetic to identify different groups. For each group, uses the <code>x</code>, <code>xmin</code>, <code>xmax</code>, and <code>thickness</code> aesthetics to draw points, intervals, and slabs.</li> </ul>

- "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 <code>c(size = FALSE)</code> , 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. <code>borders()</code> .

## 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  $[0, 1]$ .
- `level`: For intervals, the interval width as an ordered factor.
- `f`: For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by `slab_type`.
- `pdf`: For slabs, the probability density function.
- `cdf`: For slabs, the cumulative distribution function.
- `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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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	<p>The data to be displayed in this layer. There are three options:</p> <p>If <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code>.</p> <p>A <code>data.frame</code>, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p>
geom	Use to override the default connection between <code>stat_halfeye()</code> and <code>geom_slabinterval()</code>
position	Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" ( <code>position_dodge()</code> ) or "dodgejust" ( <code>position_dodgejust()</code> ) can be useful if you have overlapping geometries.
...	<p>Other arguments passed to <code>layer()</code>. These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> (see <b>Aesthetics</b>, below). They may also be parameters to the paired geom/stat. When paired with the default geom, <code>geom_slabinterval()</code>, these include:</p> <p><code>normalize</code> How to normalize heights of functions input to the thickness aesthetic. One of:</p> <ul style="list-style-type: none"> <li>• "all": normalize so that the maximum height across all data is 1.</li> <li>• "panels": normalize within panels so that the maximum height in each panel is 1.</li> <li>• "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.</li> <li>• "groups": normalize within values of the opposite axis and within each group so that the maximum height in each group is 1.</li> </ul>

- "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  $\geq$  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  $\geq$  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`.

<code>slab_type</code>	The type of slab function to calculate: probability density (or mass) function ("pdf"), cumulative distribution function ("cdf"), or complementary CDF ("ccdf").
<code>p_limits</code>	Probability limits (as a vector of size 2) used to determine the lower and upper limits of the slab. E.g., if this is <code>c(.001, .999)</code> , 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 <code>p_limits</code> is <code>c(NA, NA)</code> on a gamma distribution the effective value of <code>p_limits</code> would be <code>c(0, .999)</code> since the gamma distribution is defined on $(0, \text{Inf})$ ; whereas on a normal distribution it would be equivalent to <code>c(.001, .999)</code> since the normal distribution is defined on $(-\text{Inf}, \text{Inf})$ .
<code>adjust</code>	If <code>slab_type</code> is "pdf", bandwidth for the density estimator for sample data is adjusted by multiplying it by this value. See <code>density()</code> for more information.
<code>trim</code>	For sample data, should the density estimate be trimmed to the range of the input data? Default TRUE.
<code>expand</code>	For sample data, should the slab be expanded to the limits of the scale? Default FALSE.
<code>breaks</code>	If <code>slab_type</code> is "histogram", the breaks parameter that is passed to <code>hist()</code> to determine where to put breaks in the histogram (for sample data).
<code>outline_bars</code>	For sample data (if <code>slab_type</code> is "histogram") and for discrete analytical distributions (whose slabs are drawn as histograms), determines if outlines in between the bars are drawn when the <code>slab_color</code> 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.
<code>point_interval</code>	A function from the <code>point_interval()</code> family (e.g., <code>median_qi</code> , <code>mean_qi</code> , <code>mode_hdi</code> , 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 <code>ggdist</code> 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, hdc). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of <code>orientation</code> . See the <code>point_interval()</code> family of functions for more information.
<code>limits</code>	Manually-specified limits for the slab, as a vector of length two. These limits are combined with those computed based on <code>p_limits</code> 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. <code>limits = c(0, NA)</code> will ensure that the lower limit does not go below 0, but let the upper limit be determined by either <code>p_limits</code> or the scale settings.
<code>n</code>	Number of points at which to evaluate the function that defines the slab.
<code>.width</code>	The <code>.width</code> argument passed to <code>point_interval</code> : 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).

<code>orientation</code>	<p>Whether this geom is drawn horizontally or vertically. One of:</p> <ul style="list-style-type: none"> <li>• NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.</li> <li>• "horizontal" (or "y"): draw horizontally, using the <code>y</code> aesthetic to identify different groups. For each group, uses the <code>x</code>, <code>xmin</code>, <code>xmax</code>, and <code>thickness</code> aesthetics to draw points, intervals, and slabs.</li> <li>• "vertical" (or "x"): draw vertically, using the <code>x</code> aesthetic to identify different groups. For each group, uses the <code>y</code>, <code>ymin</code>, <code>ymax</code>, and <code>thickness</code> aesthetics to draw points, intervals, and slabs.</li> </ul> <p>For compatibility with the base <code>ggplot</code> naming scheme for orientation, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal" (tidybayes had an <code>orientation</code> parameter before base <code>ggplot</code> did, hence the discrepancy).</p>
<code>na.rm</code>	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
<code>show.legend</code>	Should this layer be included in the legends? Default is <code>c(size = FALSE)</code> , 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).
<code>inherit.aes</code>	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. <code>borders()</code> .

## 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]`.
- `level`: For intervals, the interval width as an ordered factor.
- `f`: For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by `slab_type`.
- `pdf`: For slabs, the probability density function.
- `cdf`: For slabs, the cumulative distribution function.
- `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**

```
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	<i>Histogram + interval plot (shortcut stat)</i>
-------------------	--

---

**Description**

Shortcut version of [stat\\_slabinterval\(\)](#) with [geom\\_slabinterval\(\)](#) for creating histogram + interval plots.

Roughly equivalent to:

```
stat_slabinterval(
  slab_type = "histogram"
)
```

**Usage**

```

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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
geom	Use to override the default connection between <code>stat_histinterval()</code> and <code>geom_slabinterval()</code>
position	Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" ( <code>position_dodge()</code> ) or "dodgejust" ( <code>position_dodgejust()</code> ) can be useful if you have overlapping geometries.
...	Other arguments passed to <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> (see <b>Aesthetics</b> , below). They may also be parameters to the paired geom/stat. When paired with the default geom, <code>geom_slabinterval()</code> , 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  $\geq$  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  $\geq$  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`.

<code>slab_type</code>	The type of slab function to calculate: probability density (or mass) function ("pdf"), cumulative distribution function ("cdf"), or complementary CDF ("ccdf").
<code>p_limits</code>	Probability limits (as a vector of size 2) used to determine the lower and upper limits of the slab. E.g., if this is <code>c(.001, .999)</code> , 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 <code>0.001</code> ( <code>0.999</code> ) if it is not finite. E.g., if <code>p_limits</code> is <code>c(NA, NA)</code> on a gamma distribution the effective value of <code>p_limits</code> would be <code>c(0, .999)</code> since the gamma distribution is defined on $(0, \text{Inf})$ ; whereas on a normal distribution it would be equivalent to <code>c(.001, .999)</code> since the normal distribution is defined on $(-\text{Inf}, \text{Inf})$ .
<code>adjust</code>	If <code>slab_type</code> is "pdf", bandwidth for the density estimator for sample data is adjusted by multiplying it by this value. See <a href="#">density()</a> for more information.
<code>trim</code>	For sample data, should the density estimate be trimmed to the range of the input data? Default TRUE.
<code>expand</code>	For sample data, should the slab be expanded to the limits of the scale? Default FALSE.
<code>breaks</code>	If <code>slab_type</code> is "histogram", the breaks parameter that is passed to <a href="#">hist()</a> to determine where to put breaks in the histogram (for sample data).
<code>outline_bars</code>	For sample data (if <code>slab_type</code> is "histogram") and for discrete analytical distributions (whose slabs are drawn as histograms), determines if outlines in between the bars are drawn when the <code>slab_color</code> 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.
<code>point_interval</code>	A function from the <a href="#">point_interval()</a> family (e.g., <code>median_qi</code> , <code>mean_qi</code> , <code>mode_hdi</code> , 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 <code>ggdist</code> 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, hdc). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of <code>orientation</code> . See the <a href="#">point_interval()</a> 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 <code>p_limits</code> 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 <code>NA</code> to leave a limit alone; e.g. <code>limits = c(0, NA)</code> will ensure that the lower limit does not go below 0, but let the upper limit be determined by either <code>p_limits</code> or the scale settings.
n	Number of points at which to evaluate the function that defines the slab.
.width	The <code>.width</code> argument passed to <code>point_interval</code> : 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 <code>.width</code> and <code>level</code> generated variables).
orientation	Whether this geom is drawn horizontally or vertically. One of: <ul style="list-style-type: none"> <li>• <code>NA</code> (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.</li> <li>• <code>"horizontal"</code> (or <code>"y"</code>): draw horizontally, using the <code>y</code> aesthetic to identify different groups. For each group, uses the <code>x</code>, <code>xmin</code>, <code>xmax</code>, and <code>thickness</code> aesthetics to draw points, intervals, and slabs.</li> <li>• <code>"vertical"</code> (or <code>"x"</code>): draw vertically, using the <code>x</code> aesthetic to identify different groups. For each group, uses the <code>y</code>, <code>ymin</code>, <code>ymax</code>, and <code>thickness</code> aesthetics to draw points, intervals, and slabs.</li> </ul> <p>For compatibility with the base <code>ggplot</code> naming scheme for orientation, <code>"x"</code> can be used as an alias for <code>"vertical"</code> and <code>"y"</code> as an alias for <code>"horizontal"</code> (tidybayes had an <code>orientation</code> parameter before base <code>ggplot</code> did, hence the discrepancy).</p>
na.rm	If <code>FALSE</code> , the default, missing values are removed with a warning. If <code>TRUE</code> , missing values are silently removed.
show.legend	Should this layer be included in the legends? Default is <code>c(size = FALSE)</code> , unlike most geoms, to match its common use cases. <code>FALSE</code> hides all legends, <code>TRUE</code> shows all legends, and <code>NA</code> shows only those that are mapped (the default for most geoms).
inherit.aes	If <code>FALSE</code> , 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. <code>borders()</code> .

## 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]`.
- `level`: For intervals, the interval width as an ordered factor.
- `f`: For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by `slab_type`.
- `pdf`: For slabs, the probability density function.
- `cdf`: For slabs, the cumulative distribution function.
- `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_halfeye()`, `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_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()
```

---

stat_interval	<i>Multiple-interval plot (shortcut stat)</i>
---------------	---

---

### Description

Shortcut version of `stat_slabinterval()` with `geom_interval()` for creating multiple-interval plots.

Roughly equivalent to:

```
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

```
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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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:<br>If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> .<br>A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created.<br>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ). |

geom	Use to override the default connection between <code>stat_interval()</code> and <code>geom_interval()</code>
position	Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" ( <code>position_dodge()</code> ) or "dodgejust" ( <code>position_dodgejust()</code> ) can be useful if you have overlapping geometries.
...	Other arguments passed to <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> (see <b>Aesthetics</b> , below). They may also be parameters to the paired geom/stat. When paired with the default geom, <code>geom_interval()</code> , these include: <ul style="list-style-type: none"> <li><code>interval_size_range</code> 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 <code>scale_size_continuous()</code>, which give sizes with a range of <code>c(1,6)</code>. The <code>interval_size_domain</code> value indicates the input domain of raw size values (typically this should be equal to the value of the range argument of the <code>scale_size_continuous()</code> function), and <code>interval_size_range</code> 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 <code>interval_size</code> or <code>point_size</code> aesthetics; see <b>scales</b>.</li> <li><code>interval_size_domain</code> 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 <code>interval_size_range</code> (see the documentation for that argument.)</li> </ul>
.width	The <code>.width</code> argument passed to <code>point_interval</code> : 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 <code>.width</code> and level generated variables).
point_interval	A function from the <code>point_interval()</code> family (e.g., <code>median_qi</code> , <code>mean_qi</code> , <code>mode_hdi</code> , 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 <b>ggdist</b> 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 <code>orientation</code> . See the <code>point_interval()</code> family of functions for more information.
orientation	Whether this geom is drawn horizontally or vertically. One of: <ul style="list-style-type: none"> <li>• NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.</li> <li>• "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.</li> </ul>

- "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 <code>c(size = FALSE)</code> , 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. <code>borders()</code> .

## 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  $[0, 1]$ .
- `level`: For intervals, the interval width as an ordered factor.

## 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_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:

```

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**

```
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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
geom	Use to override the default connection between <code>stat_lineribbon()</code> and <code>geom_lineribbon()</code>
position	Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" ( <code>position_dodge()</code> ) or "dodgejust" ( <code>position_dodgejust()</code> ) can be useful if you have overlapping geometries.
...	Other arguments passed to <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> (see <b>Aesthetics</b> , below). They may also be parameters to the paired geom/stat. When paired with the default geom, <code>geom_lineribbon()</code> , these include:  step Should the line/ribbon be drawn as a step function? One of: <ul style="list-style-type: none"> <li>• <code>FALSE</code> (default): do not draw as a step function.</li> <li>• <code>"mid"</code> (or <code>TRUE</code>): draw steps midway between adjacent x values.</li> <li>• <code>"hv"</code>: draw horizontal-then-vertical steps.</li> <li>• <code>"vh"</code>: draw as vertical-then-horizontal steps.</li> </ul> <code>TRUE</code> is an alias for <code>"mid"</code> because for a step function with ribbons, <code>"mid"</code> 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).

<code>.width</code>	The <code>.width</code> argument passed to <code>point_interval</code> : 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 <code>.width</code> and <code>level</code> generated variables).
<code>point_interval</code>	A function from the <code>point_interval()</code> family (e.g., <code>median_qi</code> , <code>mean_qi</code> , <code>mode_hdi</code> , etc), or a string giving the name of a function from that family (e.g., <code>"median_qi"</code> , <code>"mean_qi"</code> , <code>"mode_hdi"</code> , etc; if a string, the caller's environment is searched for the function, followed by the <code>ggdist</code> environment). This function determines the point summary (typically mean, median, or mode) and interval type (quantile interval, <code>qi</code> ; highest-density interval, <code>hdi</code> ; or highest-density continuous interval, <code>hdc</code> ). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of <code>orientation</code> . See the <code>point_interval()</code> family of functions for more information.
<code>orientation</code>	Whether this geom is drawn horizontally or vertically. One of: <ul style="list-style-type: none"> <li>• <code>NA</code> (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.</li> <li>• <code>"horizontal"</code> (or <code>"y"</code>): draw horizontally, using the <code>y</code> aesthetic to identify different groups. For each group, uses the <code>x</code>, <code>xmin</code>, <code>xmax</code>, and <code>thickness</code> aesthetics to draw points, intervals, and slabs.</li> <li>• <code>"vertical"</code> (or <code>"x"</code>): draw vertically, using the <code>x</code> aesthetic to identify different groups. For each group, uses the <code>y</code>, <code>ymin</code>, <code>ymax</code>, and <code>thickness</code> aesthetics to draw points, intervals, and slabs.</li> </ul> <p>For compatibility with the base <code>ggplot</code> naming scheme for orientation, <code>"x"</code> can be used as an alias for <code>"vertical"</code> and <code>"y"</code> as an alias for <code>"horizontal"</code> (tidybayes had an <code>orientation</code> parameter before base <code>ggplot</code> did, hence the discrepancy).</p>
<code>na.rm</code>	If <code>FALSE</code> , the default, missing values are removed with a warning. If <code>TRUE</code> , missing values are silently removed.
<code>show.legend</code>	Should this layer be included in the legends? <code>NA</code> , the default, includes if any aesthetics are mapped. <code>FALSE</code> never includes, and <code>TRUE</code> always includes.
<code>inherit.aes</code>	If <code>FALSE</code> , 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. <code>borders()</code> .

## 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 line + 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]`.
- `level`: For intervals, the interval width as an ordered factor.

## 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.

### 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\(\)](#)

### Examples

```
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:

```

stat_slabinterval(
  geom = "pointinterval",
  show_slab = FALSE
)

```

### Usage

```

stat_pointinterval(
  mapping = NULL,
  data = NULL,
  geom = "pointinterval",
  position = "identity",
  ...,
  point_interval = "median_qi",
  .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	<p>The data to be displayed in this layer. There are three options:</p> <p>If NULL, the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code>.</p> <p>A <code>data.frame</code>, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p>
geom	<p>Use to override the default connection between <code>stat_pointinterval()</code> and <code>geom_pointinterval()</code></p>
position	<p>Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" (<code>position_dodge()</code>) or "dodgejust" (<code>position_dodgejust()</code>) can be useful if you have overlapping geometries.</p>
...	<p>Other arguments passed to <code>layer()</code>. These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> (see <b>Aesthetics</b>, below). They may also be parameters to the paired geom/stat. When paired with the default geom, <code>geom_pointinterval()</code>, these include:</p>
	<p><code>interval_size_domain</code> 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 <code>interval_size_range</code> (see the documentation for that argument.)</p>
	<p><code>interval_size_range</code> 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 <code>scale_size_continuous()</code>, which give sizes with a range of <code>c(1, 6)</code>. The <code>interval_size_domain</code> value indicates the input domain of raw size values (typically this should be equal to the value of the range argument of the <code>scale_size_continuous()</code> function), and <code>interval_size_range</code> 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 <code>interval_size</code> or <code>point_size</code> aesthetics; see <b>scales</b>.</p>
	<p><code>fatten_point</code> 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 <code>point_size</code> aesthetic and <code>scale_point_size_continuous()</code> or <code>scale_point_size_discrete()</code>; sizes specified with that aesthetic will not be adjusted using <code>fatten_point</code>.</p>
point_interval	<p>A function from the <code>point_interval()</code> family (e.g., <code>median_qi</code>, <code>mean_qi</code>, <code>mode_hdi</code>, 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 <b>ggdist</b> environment). This function determines the point summary (typically mean, median, or mode) and interval</p>

	type (quantile interval, <code>qi</code> ; highest-density interval, <code>hdi</code> ; or highest-density continuous interval, <code>hdci</code> ). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of <code>orientation</code> . See the <a href="#">point_interval()</a> family of functions for more information.
<code>.width</code>	The <code>.width</code> argument passed to <code>point_interval</code> : 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 <code>.width</code> and level generated variables).
<code>orientation</code>	Whether this geom is drawn horizontally or vertically. One of: <ul style="list-style-type: none"> <li>• NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.</li> <li>• "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify different groups. For each group, uses the x, <code>xmin</code>, <code>xmax</code>, and <code>thickness</code> aesthetics to draw points, intervals, and slabs.</li> <li>• "vertical" (or "x"): draw vertically, using the x aesthetic to identify different groups. For each group, uses the y, <code>ymin</code>, <code>ymax</code>, and <code>thickness</code> aesthetics to draw points, intervals, and slabs.</li> </ul> <p>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).</p>
<code>na.rm</code>	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
<code>show.legend</code>	Should this layer be included in the legends? Default is <code>c(size = FALSE)</code> , 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).
<code>inherit.aes</code>	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. <a href="#">borders()</a> .

## 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]`.
- `level`: For intervals, the interval width as an ordered factor.

## 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 **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.
- `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_pointinterval()` 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_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_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	<i>Multiple-ribbon plot (shortcut stat)</i>
-------------	---

---

### 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:

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

geom	Use to override the default connection between <code>stat_ribbon()</code> and <code>geom_lineribbon()</code>
position	Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" ( <code>position_dodge()</code> ) or "dodgejust" ( <code>position_dodgejust()</code> ) can be useful if you have overlapping geometries.
...	Other arguments passed to <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> (see <b>Aesthetics</b> , below). They may also be parameters to the paired geom/stat. When paired with the default geom, <code>geom_lineribbon()</code> , these include: <ul style="list-style-type: none"> <li>step Should the line/ribbon be drawn as a step function? One of: <ul style="list-style-type: none"> <li>FALSE (default): do not draw as a step function.</li> <li>"mid" (or TRUE): draw steps midway between adjacent x values.</li> <li>"hv": draw horizontal-then-vertical steps.</li> <li>"vh": draw as vertical-then-horizontal steps.</li> </ul> <p>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).</p> </li> </ul>
.width	The <code>.width</code> argument passed to <code>point_interval</code> : 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 <code>.width</code> and <code>level</code> generated variables).
point_interval	A function from the <code>point_interval()</code> family (e.g., <code>median_qi</code> , <code>mean_qi</code> , <code>mode_hdi</code> , 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 <b>ggdist</b> 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, hdc). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of <code>orientation</code> . See the <code>point_interval()</code> family of functions for more information.
orientation	Whether this geom is drawn horizontally or vertically. One of: <ul style="list-style-type: none"> <li>NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.</li> <li>"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.</li> <li>"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.</li> </ul> <p>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).</p>
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. <code>borders()</code> .

## 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]`.
- `level`: For intervals, the interval width as an ordered factor.

## 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**

```

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_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()

```

---

stat\_slab

*Slab (ridge) plot (shortcut stat)*


---

**Description**

Shortcut version of `stat_slabinterval()` with `geom_slab()` for creating slab (ridge) plots.

Roughly equivalent to:

```

stat_slabinterval(
  aes(size = NULL),
  geom = "slab",
  show.legend = NA
)

```

**Usage**

```

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

```

### Arguments

mapping	Set of aesthetic mappings created by <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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	<p>The data to be displayed in this layer. There are three options:</p> <p>If <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code>.</p> <p>A <code>data.frame</code>, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p>
geom	Use to override the default connection between <code>stat_slab()</code> and <code>geom_slab()</code>
position	Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" ( <code>position_dodge()</code> ) or "dodgejust" ( <code>position_dodgejust()</code> ) can be useful if you have overlapping geometries.
...	<p>Other arguments passed to <code>layer()</code>. These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> (see <b>Aesthetics</b>, below). They may also be parameters to the paired geom/stat. When paired with the default geom, <code>geom_slab()</code>, these include:</p> <p><code>normalize</code> How to normalize heights of functions input to the thickness aesthetic. One of:</p> <ul style="list-style-type: none"> <li>• "all": normalize so that the maximum height across all data is 1.</li> <li>• "panels": normalize within panels so that the maximum height in each panel is 1.</li> <li>• "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.</li> <li>• "groups": normalize within values of the opposite axis and within each group so that the maximum height in each group is 1.</li> </ul>



- "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  $R \geq 4.1$  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  $R \geq 4.2$ , 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.

<code>slab_type</code>	The type of slab function to calculate: probability density (or mass) function ("pdf"), cumulative distribution function ("cdf"), or complementary CDF ("ccdf").
<code>p_limits</code>	Probability limits (as a vector of size 2) used to determine the lower and upper limits of the slab. E.g., if this is <code>c(.001, .999)</code> , 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 <code>p_limits</code> is <code>c(NA, NA)</code> on a gamma distribution the effective value of <code>p_limits</code> would be <code>c(0, .999)</code> since the gamma distribution is defined on $(0, \text{Inf})$ ; whereas on a normal distribution it would be equivalent to <code>c(.001, .999)</code> since the normal distribution is defined on $(-\text{Inf}, \text{Inf})$ .
<code>adjust</code>	If <code>slab_type</code> is "pdf", bandwidth for the density estimator for sample data is adjusted by multiplying it by this value. See <code>density()</code> for more information.
<code>trim</code>	For sample data, should the density estimate be trimmed to the range of the input data? Default TRUE.
<code>expand</code>	For sample data, should the slab be expanded to the limits of the scale? Default FALSE.
<code>breaks</code>	If <code>slab_type</code> is "histogram", the breaks parameter that is passed to <code>hist()</code> 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: <ul style="list-style-type: none"> <li>• NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.</li> <li>• "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.</li> <li>• "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.</li> </ul> <p>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).</p>
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` 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]`.
- `level`: For intervals, the interval width as an ordered factor.
- `f`: For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by `slab_type`.
- `pdf`: For slabs, the probability density function.
- `cdf`: For slabs, the cumulative distribution function.
- `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

```
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	<i>Slab + interval plots for sample data and analytical distributions (ggplot stat)</i>
-------------------	---

---

## 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",
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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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	<p>The data to be displayed in this layer. There are three options:</p> <p>If <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code>.</p> <p>A <code>data.frame</code>, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p>
geom	Use to override the default connection between <code>stat_slabinterval()</code> and <code>geom_slabinterval()</code>
position	Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" ( <code>position_dodge()</code> ) or "dodgejust" ( <code>position_dodgejust()</code> ) can be useful if you have overlapping geometries.
...	<p>Other arguments passed to <code>layer()</code>. These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> (see <b>Aesthetics</b>, below). They may also be parameters to the paired geom/stat. When paired with the default geom, <code>geom_slabinterval()</code>, these include:</p> <p><code>normalize</code> How to normalize heights of functions input to the thickness aesthetic. One of:</p> <ul style="list-style-type: none"> <li>• "all": normalize so that the maximum height across all data is 1.</li> <li>• "panels": normalize within panels so that the maximum height in each panel is 1.</li> </ul>

- "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 <code>interval_size</code> or <code>point_size</code> aesthetics; see <a href="#">scales</a> .
	<code>fatten_point</code> 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 <code>point_size</code> aesthetic and <a href="#">scale_point_size_continuous()</a> or <a href="#">scale_point_size_discrete()</a> ; sizes specified with that aesthetic will not be adjusted using <code>fatten_point</code> .
<code>slab_type</code>	The type of slab function to calculate: probability density (or mass) function ("pdf"), cumulative distribution function ("cdf"), or complementary CDF ("ccdf").
<code>p_limits</code>	Probability limits (as a vector of size 2) used to determine the lower and upper limits of the slab. E.g., if this is <code>c(.001, .999)</code> , 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 <code>p_limits</code> is <code>c(NA, NA)</code> on a gamma distribution the effective value of <code>p_limits</code> would be <code>c(0, .999)</code> since the gamma distribution is defined on $(0, \text{Inf})$ ; whereas on a normal distribution it would be equivalent to <code>c(.001, .999)</code> since the normal distribution is defined on $(-\text{Inf}, \text{Inf})$ .
<code>adjust</code>	If <code>slab_type</code> is "pdf", bandwidth for the density estimator for sample data is adjusted by multiplying it by this value. See <a href="#">density()</a> for more information.
<code>trim</code>	For sample data, should the density estimate be trimmed to the range of the input data? Default TRUE.
<code>expand</code>	For sample data, should the slab be expanded to the limits of the scale? Default FALSE.
<code>breaks</code>	If <code>slab_type</code> is "histogram", the <code>breaks</code> parameter that is passed to <a href="#">hist()</a> to determine where to put breaks in the histogram (for sample data).
<code>outline_bars</code>	For sample data (if <code>slab_type</code> is "histogram") and for discrete analytical distributions (whose slabs are drawn as histograms), determines if outlines in between the bars are drawn when the <code>slab_color</code> 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.
<code>point_interval</code>	A function from the <a href="#">point_interval()</a> family (e.g., <code>median_qi</code> , <code>mean_qi</code> , <code>mode_hdi</code> , 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 <code>ggdist</code> 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, hdc). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of <code>orientation</code> . See the <a href="#">point_interval()</a> family of functions for more information.
<code>limits</code>	Manually-specified limits for the slab, as a vector of length two. These limits are combined with those computed based on <code>p_limits</code> 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. <code>limits = c(0, NA)</code> will ensure that the lower limit does not go below 0, but let the upper limit be determined by either <code>p_limits</code> 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: <ul style="list-style-type: none"> <li>• NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.</li> <li>• "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.</li> <li>• "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.</li> </ul> <p>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).</p>
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. <code>borders()</code> .

## 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  $[0, 1]$ .
- `level`: For intervals, the interval width as an ordered factor.
- `f`: For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by `slab_type`.
- `pdf`: For slabs, the probability density function.
- `cdf`: For slabs, the cumulative distribution function.
- `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 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

```
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):
df %>%
  ggplot(aes(x = group, y = value)) +
  stat_eye(aes(thickness = stat(pdf*n)))

# EXAMPLES ON ANALYTICAL DISTRIBUTIONS

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
)

# Using functions from the distributional package (like dist_normal()) with the
# dist aesthetic can lead to more compact/expressive specifications

dist_df %>%
  ggplot(aes(x = group, ydist = dist_normal(mean, sd), fill = subgroup)) +
  stat_eye(position = "dodge")

# using the old character vector + args approach
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:
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)
```

```
rstudent_t(n, df, mu = 0, sigma = 1)
```

### Arguments

x	vector of quantiles.
df	degrees of freedom ( $> 0$ , maybe non-integer). $df = \text{Inf}$ is allowed.
mu	Location parameter (median)
sigma	Scale parameter
log	logical; if TRUE, probabilities p are given as $\log(p)$ .
q	vector of quantiles.
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ .
log.p	logical; if TRUE, probabilities p are given as $\log(p)$ .
p	vector of probabilities.
n	number of observations. If $\text{length}(n) > 1$ , 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**

```
library(ggplot2)

theme_set(theme_ggdist())
```

---

tidy-format-translators

*Translate between different tidy data frame formats for draws from distributions*

---

**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**

```
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

```
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

df %>%
  to_broom_names()
```

# Index

- \* **datasets**
  - ggdist-deprecated, 44
- \* **ggdist scales**
  - scale\_colour\_ramp, 63
  - scales, 58
- \* **lineribbon stats**
  - stat\_lineribbon, 120
  - stat\_ribbon, 131
- \* **manip**
  - tidy-format-translators, 154
- \* **slabinterval geoms**
  - geom\_interval, 21
  - geom\_pointinterval, 28
  - geom\_slab, 33
- \* **slabinterval stats**
  - stat\_ccdfinterval, 65
  - stat\_cdfinterval, 73
  - stat\_eye, 82
  - stat\_gradientinterval, 90
  - stat\_halfeye, 98
  - stat\_histinterval, 106
  - stat\_interval, 115
  - stat\_pointinterval, 125
  - stat\_slab, 135
- aes(), 13, 17, 21, 25, 29, 33, 38, 66, 69, 70, 74, 77, 78, 82, 86, 91, 94, 99, 102, 103, 107, 111, 115, 117, 121–123, 125, 127, 128, 131, 133, 136, 139, 143, 147
- aes\_(), 13, 21, 25, 29, 33, 38, 66, 74, 82, 91, 99, 107, 115, 121, 125, 131, 136, 143
- after\_stat(), 70, 78, 86, 94, 103, 111, 117, 123, 128, 133, 139, 147
- axis\_titles\_bottom\_left(theme\_ggdist), 153
- beeswarm::beeswarm(), 4, 14
- bin\_dots, 4
- bin\_dots(), 11
- borders(), 16, 22, 26, 30, 35, 40, 69, 77, 86, 94, 102, 110, 117, 122, 127, 133, 138, 146
- cdf(), 9
- continuous\_scale(), 64
- coord\_cartesian(), 61, 64
- curve\_interval, 6
- curve\_interval(), 6
- cut\_cdf\_qi, 9
- cut\_cdf\_qi(), 9
- density(), 68, 76, 84, 93, 101, 109, 137, 145
- discrete\_scale(), 64
- dist\_beta(), 17, 69, 77, 86, 94, 102, 110, 117, 122, 127, 133, 138, 147
- dist\_normal(), 17, 69, 70, 77–79, 86, 87, 94, 95, 102, 103, 110, 112, 117, 118, 122, 123, 127, 128, 133, 134, 138–140, 147, 148
- dlkcorr\_marginal(lkcorr\_marginal), 45
- dnorm(), 17, 69, 78, 86, 94, 102, 111, 117, 123, 128, 133, 139, 147
- dplyr::filter(), 48
- dplyr::group\_by(), 55
- dplyr::select(), 6
- dstudent\_t(student\_t), 152
- facet\_title\_horizontal(theme\_ggdist), 153
- facet\_title\_left\_horizontal(theme\_ggdist), 153
- facet\_title\_right\_horizontal(theme\_ggdist), 153
- fda::fbplot(), 7
- find\_dotplot\_binwidth, 10
- find\_dotplot\_binwidth(), 5
- fortify(), 13, 21, 26, 29, 33, 38, 66, 74, 83, 91, 99, 107, 115, 121, 126, 131, 136, 143

- from\_broom\_names  
(tidy-format-translators), 154
- from\_broom\_names(), 155
- from\_ggmcmc\_names  
(tidy-format-translators), 154
- from\_ggmcmc\_names(), 155
- geom\_dotplot(), 16
- geom\_dots (geom\_dotsinterval), 12
- geom\_dotsinterval, 12
- geom\_dotsinterval(), 3, 5, 11, 17, 44, 62
- geom\_interval, 21, 32, 37
- geom\_interval(), 23, 41, 115, 116, 118, 119
- geom\_line(), 25, 27, 28
- geom\_lineribbon, 25
- geom\_lineribbon(), 3, 27, 44, 63, 120, 121, 124, 131, 132, 134
- geom\_pointinterval, 24, 28, 37
- geom\_pointinterval(), 28, 30, 41, 125, 126, 128, 130
- geom\_ribbon(), 25, 27, 28
- geom\_slab, 24, 32, 33
- geom\_slab(), 41, 135, 136, 140, 141
- geom\_slabinterval, 38
- geom\_slabinterval(), 3, 5, 12, 16, 18, 20, 21, 23, 24, 28, 30, 32, 33, 36, 37, 40–42, 45, 56, 58, 61, 63, 65, 66, 70, 71, 73, 74, 79, 81–83, 87, 89–91, 95–97, 99, 103, 104, 106, 107, 112, 114, 140, 142, 143, 148, 150
- ggdist (ggdist-package), 3
- ggdist-deprecated, 44
- ggdist-package, 3
- ggplot(), 13, 17, 21, 23, 25, 27, 29, 30, 33, 35, 38, 41, 62, 64, 66, 69, 74, 78, 82, 86, 91, 94, 99, 103, 107, 111, 115, 117, 121, 123, 126, 128, 131, 133, 136, 139, 143, 147
- ggplot2, 23, 30
- ggplot2::Geom, 17, 23, 27, 30, 35, 41
- ggplot2::position\_dodge(), 56
- ggplot2::Scale, 62, 64
- ggplot2::Stat, 17, 69, 78, 86, 94, 103, 111, 117, 123, 128, 133, 139, 147
- ggplot2::theme(), 154
- ggplot2::theme\_light(), 154
- ggplot2::theme\_set(), 154
- grob, 11
- group\_by(), 6, 54
- guide\_colorbar2 (scales), 58
- guide\_colourbar2 (scales), 58
- hdci (point\_interval), 51
- hdi (point\_interval), 51
- hdi(), 54
- HDInterval::hdi(), 55
- hist(), 68, 76, 84, 93, 101, 109, 137, 145
- lambda, 61, 64
- layer(), 14, 22, 26, 29, 34, 39, 66, 74, 83, 91, 99, 107, 116, 121, 126, 132, 136, 143
- lkjcorr\_marginal, 45
- lkjcorr\_marginal(), 47, 48
- ll (point\_interval), 51
- make.names(), 50
- marginalize\_lkjcorr, 47
- marginalize\_lkjcorr(), 47
- mean(), 54
- mean\_hdci (point\_interval), 51
- mean\_hdi (point\_interval), 51
- mean\_ll (point\_interval), 51
- mean\_qi (point\_interval), 51
- mean\_qi(), 23, 30
- mean\_ul (point\_interval), 51
- median(), 54
- median\_hdci (point\_interval), 51
- median\_hdi (point\_interval), 51
- median\_ll (point\_interval), 51
- median\_qi (point\_interval), 51
- median\_qi(), 23, 30
- median\_ul (point\_interval), 51
- Mode (point\_interval), 51
- Mode(), 54
- mode\_hdci (point\_interval), 51
- mode\_hdi (point\_interval), 51
- mode\_hdi(), 23, 30
- mode\_ll (point\_interval), 51
- mode\_qi (point\_interval), 51
- mode\_ul (point\_interval), 51
- ordered, 9
- parse\_dist, 49
- parse\_dist(), 17, 47, 48, 50, 69, 78, 86, 94, 102, 111, 117, 123, 128, 133, 139, 147, 153
- plkjcorr\_marginal (lkjcorr\_marginal), 45

- pnorm(), [9](#), [17](#), [69](#), [78](#), [86](#), [94](#), [102](#), [111](#), [117](#),  
[123](#), [128](#), [133](#), [139](#), [147](#)
- point\_interval, [51](#)
- point\_interval(), [8](#), [16](#), [23](#), [25](#), [27](#), [30](#), [68](#),  
[76](#), [85](#), [93](#), [101](#), [109](#), [116](#), [120](#), [122](#),  
[126](#), [127](#), [131](#), [132](#), [145](#)
- position\_dodge(), [14](#), [22](#), [29](#), [34](#), [39](#), [66](#), [74](#),  
[83](#), [91](#), [99](#), [107](#), [116](#), [121](#), [126](#), [132](#),  
[136](#), [143](#)
- position\_dodgejust, [56](#)
- position\_dodgejust(), [14](#), [22](#), [29](#), [34](#), [39](#),  
[56](#), [66](#), [74](#), [83](#), [91](#), [99](#), [107](#), [116](#), [121](#),  
[126](#), [132](#), [136](#), [143](#)
- posterior::rvar(), [17](#), [69](#), [70](#), [77–79](#), [86](#),  
[87](#), [94](#), [95](#), [102](#), [103](#), [110](#), [112](#), [117](#),  
[118](#), [122](#), [123](#), [127](#), [128](#), [133](#), [134](#),  
[138–140](#), [147](#), [148](#)
- pstudent\_t(student\_t), [152](#)
- qi(point\_interval), [51](#)
- qi(), [54](#)
- qlkcorr\_marginal(lkcorr\_marginal), [45](#)
- qnorm(), [17](#), [69](#), [78](#), [86](#), [94](#), [102](#), [111](#), [117](#),  
[123](#), [128](#), [133](#), [139](#), [147](#)
- qstudent\_t(student\_t), [152](#)
- r\_dist\_name(parse\_dist), [49](#)
- r\_dist\_name(), [50](#)
- rlang::eval\_tidy(), [54](#)
- rlkcorr\_marginal(lkcorr\_marginal), [45](#)
- rstudent\_t(student\_t), [152](#)
- scale\_alpha\_continuous(), [61](#)
- scale\_color\_continuous(), [62](#)
- scale\_color\_discrete(), [61](#), [62](#)
- scale\_color\_ramp(scale\_colour\_ramp), [63](#)
- scale\_color\_ramp\_continuous  
(scale\_colour\_ramp), [63](#)
- scale\_color\_ramp\_discrete  
(scale\_colour\_ramp), [63](#)
- scale\_colour\_ramp, [62](#), [63](#)
- scale\_colour\_ramp(), [18](#), [23](#), [31](#), [36](#), [42](#), [71](#),  
[80](#), [88](#), [96](#), [104](#), [113](#), [118](#), [129](#), [140](#),  
[149](#)
- scale\_colour\_ramp\_continuous  
(scale\_colour\_ramp), [63](#)
- scale\_colour\_ramp\_continuous(), [64](#)
- scale\_colour\_ramp\_discrete  
(scale\_colour\_ramp), [63](#)
- scale\_colour\_ramp\_discrete(), [64](#)
- scale\_fill\_ramp(scale\_colour\_ramp), [63](#)
- scale\_fill\_ramp(), [18](#), [23](#), [27](#), [31](#), [36](#), [42](#),  
[71](#), [80](#), [88](#), [96](#), [104](#), [113](#), [119](#), [124](#),  
[129](#), [134](#), [140](#), [149](#)
- scale\_fill\_ramp\_continuous  
(scale\_colour\_ramp), [63](#)
- scale\_fill\_ramp\_discrete  
(scale\_colour\_ramp), [63](#)
- scale\_interval\_alpha\_continuous  
(scales), [58](#)
- scale\_interval\_alpha\_discrete(scales),  
[58](#)
- scale\_interval\_color\_continuous  
(scales), [58](#)
- scale\_interval\_color\_discrete(scales),  
[58](#)
- scale\_interval\_colour\_continuous  
(scales), [58](#)
- scale\_interval\_colour\_discrete  
(scales), [58](#)
- scale\_interval\_linetype\_continuous  
(scales), [58](#)
- scale\_interval\_linetype\_discrete  
(scales), [58](#)
- scale\_interval\_size\_continuous  
(scales), [58](#)
- scale\_interval\_size\_discrete(scales),  
[58](#)
- scale\_point\_alpha\_continuous(scales),  
[58](#)
- scale\_point\_alpha\_discrete(scales), [58](#)
- scale\_point\_color\_continuous(scales),  
[58](#)
- scale\_point\_color\_discrete(scales), [58](#)
- scale\_point\_colour\_continuous(scales),  
[58](#)
- scale\_point\_colour\_discrete(scales), [58](#)
- scale\_point\_fill\_continuous(scales), [58](#)
- scale\_point\_fill\_discrete(scales), [58](#)
- scale\_point\_size\_continuous(scales), [58](#)
- scale\_point\_size\_continuous(), [15](#), [30](#),  
[40](#), [67](#), [75](#), [84](#), [92](#), [100](#), [109](#), [126](#), [145](#)
- scale\_point\_size\_discrete(scales), [58](#)
- scale\_point\_size\_discrete(), [15](#), [30](#), [40](#),  
[67](#), [75](#), [84](#), [92](#), [101](#), [109](#), [126](#), [145](#)
- scale\_size\_continuous(), [15](#), [22](#), [30](#), [40](#),  
[41](#), [67](#), [75](#), [84](#), [92](#), [100](#), [108](#), [116](#),

- 126, 144*
- scale\_slab\_alpha\_continuous (scales), [58](#)
- scale\_slab\_alpha\_discrete (scales), [58](#)
- scale\_slab\_color\_continuous (scales), [58](#)
- scale\_slab\_color\_discrete (scales), [58](#)
- scale\_slab\_colour\_continuous (scales), [58](#)
- scale\_slab\_colour\_discrete (scales), [58](#)
- scale\_slab\_fill\_continuous (scales), [58](#)
- scale\_slab\_fill\_discrete (scales), [58](#)
- scale\_slab\_linetype\_continuous (scales), [58](#)
- scale\_slab\_linetype\_discrete (scales), [58](#)
- scale\_slab\_shape\_continuous (scales), [58](#)
- scale\_slab\_shape\_discrete (scales), [58](#)
- scale\_slab\_size\_continuous (scales), [58](#)
- scale\_slab\_size\_discrete (scales), [58](#)
- scales, [15](#), [19](#), [22](#), [24](#), [27](#), [30](#), [32](#), [37](#), [40](#), [43](#), [58](#), [64](#), [67](#), [72](#), [75](#), [81](#), [84](#), [89](#), [92](#), [97](#), [100](#), [105](#), [109](#), [114](#), [116](#), [119](#), [124](#), [126](#), [130](#), [134](#), [141](#), [145](#), [150](#)
- scales::percent\_format(), [9](#)
- stat(), [70](#), [78](#), [86](#), [94](#), [103](#), [111](#), [117](#), [123](#), [128](#), [133](#), [139](#), [147](#)
- stat\_ccdfinterval, [65](#), [81](#), [89](#), [97](#), [106](#), [114](#), [119](#), [130](#), [141](#)
- stat\_ccdfinterval(), [66](#), [146](#)
- stat\_cdfinterval, [73](#), [73](#), [89](#), [97](#), [106](#), [114](#), [119](#), [130](#), [141](#)
- stat\_cdfinterval(), [74](#), [146](#)
- stat\_dist\_ccdfinterval (ggdist-deprecated), [44](#)
- stat\_dist\_cdfinterval (ggdist-deprecated), [44](#)
- stat\_dist\_dots (ggdist-deprecated), [44](#)
- stat\_dist\_dotsinterval (ggdist-deprecated), [44](#)
- stat\_dist\_eye (ggdist-deprecated), [44](#)
- stat\_dist\_gradientinterval (ggdist-deprecated), [44](#)
- stat\_dist\_halfeye (ggdist-deprecated), [44](#)
- stat\_dist\_interval (ggdist-deprecated), [44](#)
- stat\_dist\_lineribbon (ggdist-deprecated), [44](#)
- stat\_dist\_pointinterval (ggdist-deprecated), [44](#)
- stat\_dist\_slab (ggdist-deprecated), [44](#)
- stat\_dist\_slabinterval (ggdist-deprecated), [44](#)
- stat\_dots (geom\_dotsinterval), [12](#)
- stat\_dots(), [12](#), [16](#)
- stat\_dotsinterval (geom\_dotsinterval), [12](#)
- stat\_dotsinterval(), [3](#), [12](#), [16](#)
- stat\_eye, [73](#), [81](#), [82](#), [97](#), [106](#), [114](#), [119](#), [130](#), [141](#)
- stat\_eye(), [55](#), [83](#), [146](#)
- stat\_gradientinterval, [73](#), [81](#), [89](#), [90](#), [106](#), [114](#), [119](#), [130](#), [141](#)
- stat\_gradientinterval(), [34](#), [39](#), [61](#), [66](#), [75](#), [83](#), [91](#), [92](#), [100](#), [108](#), [137](#), [144](#), [146](#)
- stat\_halfeye, [73](#), [81](#), [89](#), [97](#), [98](#), [114](#), [119](#), [130](#), [141](#)
- stat\_halfeye(), [9](#), [44](#), [49](#), [55](#), [99](#), [146](#)
- stat\_histinterval, [73](#), [81](#), [89](#), [97](#), [106](#), [106](#), [119](#), [130](#), [141](#)
- stat\_histinterval(), [107](#), [146](#)
- stat\_interval, [73](#), [81](#), [89](#), [97](#), [106](#), [114](#), [115](#), [130](#), [141](#)
- stat\_interval(), [24](#), [116](#), [147](#)
- stat\_lineribbon, [120](#), [134](#)
- stat\_lineribbon(), [3](#), [28](#), [120](#), [121](#)
- stat\_pointinterval, [73](#), [81](#), [89](#), [97](#), [106](#), [114](#), [119](#), [125](#), [141](#)
- stat\_pointinterval(), [32](#), [126](#), [147](#)
- stat\_ribbon, [124](#), [131](#)
- stat\_ribbon(), [131](#), [132](#)
- stat\_sample\_slabinterval (ggdist-deprecated), [44](#)
- stat\_slab, [73](#), [81](#), [89](#), [97](#), [106](#), [114](#), [119](#), [130](#), [135](#)
- stat\_slab(), [37](#), [136](#), [146](#)
- stat\_slabinterval, [142](#)
- stat\_slabinterval(), [3](#), [9](#), [12](#), [16](#), [20](#), [41](#), [44](#), [45](#), [47–49](#), [51](#), [65](#), [73](#), [81](#), [82](#), [89](#), [90](#), [97](#), [98](#), [106](#), [114](#), [115](#), [119](#), [120](#), [125](#), [130](#), [131](#), [135](#), [141](#), [143](#), [153](#)
- stat\_summary(), [55](#)
- StatDistSlabinterval (ggdist-deprecated), [44](#)
- StatSampleSlabinterval (ggdist-deprecated), [44](#)

student\_t, [152](#)

theme\_ggdist, [153](#)  
theme\_ggdist(), [154](#)  
theme\_tidybayes (theme\_ggdist), [153](#)  
theme\_tidybayes(), [154](#)  
tidy-format-translators, [154](#)  
to\_broom\_names  
    (tidy-format-translators), [154](#)  
to\_broom\_names(), [155](#)  
to\_ggmcmc\_names  
    (tidy-format-translators), [154](#)  
to\_ggmcmc\_names(), [155](#)

ul (point\_interval), [51](#)  
unit, [14](#)  
unit(), [14](#)