# Package 'ggdist'

July 12, 2020

**Title** Visualizations of Distributions and Uncertainty

**Version** 2.2.0

**Date** 2020-07-11

**Maintainer** Matthew Kay <mjskay@umich.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) <doi:10.1111/1467-985X.00120>,
density plots, gradient plots, dot plots (Wilkinson L., 1999) <doi:10.1080/00031305.1999.10474474>,
quantile dot plots (Kay M., Kola T., Hullman J., Munson S., 2016) <doi:10.1145/2858036.2858558>,
complementary cumulative distribution function
barplots (Fernandes M., Walls L., Munson S., Hullman J., Kay M., 2018) <doi:10.1145/3173574.3173718>,
and fit curves with multiple uncertainty ribbons.

**Depends** R (>= 3.5.0)

**Imports** dplyr (>= 0.8.0), tidyr (>= 1.0.0), ggplot2 (>= 3.3.0), purrr
(>= 0.2.3), rlang (>= 0.3.0), scales, grid, forcats,
HDInterval, tibble, vctrs, distributional

**Suggests** knitr, testthat, vdiffr (>= 0.3.0), svglite, broom (>=
0.5.6), modelr, cowplot, covr, gdtools, rmarkdown, png, pkgdown

**License** GPL (>= 3)

**Language** en-US

**BugReports** https://github.com/mjskay/ggdist/issues/new

**URL** http://mjskay.github.io/ggdist, https://github.com/mjskay/ggdist

1

**VignetteBuilder** knitr

**RoxygenNote** 7.1.1

**LazyData** true

**Encoding** UTF-8

**Collate** ``ggdist-package.R'' ``util.R'' ``binning_methods.R'' ``data.R''
    ``draw_key_slabinterval.R'' ``geom.R'' ``geom_slabinterval.R''
    ``geom_dotsinterval.R'' ``geom_interval.R'' ``geom_lineribbon.R''
    ``geom_pointinterval.R'' ``lkjcorr_marginal.R'' ``parse_dist.R''
    ``point_interval.R'' ``scales.R'' ``stat.R'' ``stat_slabinterval.R''
    ``stat_dist_slabinterval.R'' ``stat_sample_slabinterval.R''
    ``stat_dotsinterval.R'' ``stat_pointinterval.R'' ``stat_interval.R''
    ``stat_lineribbon.R'' ``student_t.R'' ``theme_ggdist.R''
    ``tidy_format_translators.R''

**NeedsCompilation** no

**Author** Matthew Kay [aut, cre]

**Repository** CRAN

**Date/Publication** 2020-07-12 05:30:02 UTC

# R **topics documented:**

---

| ggdist-package | *Visualizations of Distributions and Uncertainty* |

---

### Description

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

### Details

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

The [geom_slabinterval()](#) / [stat_slabinterval()](#) / [stat_dist_slabinterval()](#) family (see vignette("slabinterval")) includes point summaries and intervals, eye plots, half-eye plots, CCDF bar plots, gradient plots, dotplots, and histograms.

The [geom_lineribbon()](#) / [stat_lineribbon()](#) / [stat_dist_lineribbon()](#) family makes it easy to visualize fit lines with an arbitrary number of uncertainty bands.

---

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

---

### Description

Geoms and stats for creating dotplots that automatically determines a bin width that ensures the plot fits within the available space. Also ensures dots do not overlap, and allows generation of quantile dotplots using the quantiles argument to stat_dotsinterval/stat_dots and stat_dist_dotsinterval/stat_dist_dot 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",
  ...,
  dotsize = 1,
  stackratio = 1,
```

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

geom_dots(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  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,
  na.rm = FALSE,
  show.legend = c(size = FALSE),
  inherit.aes = TRUE
)

stat_dots(
  mapping = NULL,
  data = NULL,
  geom = "dots",
  position = "identity",
  ...,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_dist_dotsinterval(
  mapping = NULL,
  data = NULL,
  geom = "dotsinterval",
  position = "identity",
  ...,
  quantiles = 100,
  na.rm = FALSE,
```

```
    show.legend = c(size = FALSE),
    inherit.aes = TRUE
)

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

## Arguments

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

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

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

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

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

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

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

...        Arguments passed on to [geom_slabinterval](#)

        side Which side to draw 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.

        orientation Whether this geom is drawn horizontally (`"horizontal"`) or vertically (`"vertical"`). The default, `NA`, automatically detects the orientation based on how the aesthetics are assigned, and should generally do an okay job at this. When horizontal (resp. vertical), the geom uses the y (resp. x)

aesthetic to identify different groups, then for each group uses the x (resp. y) aesthetic and the thickness aesthetic to draw a function as an slab, and draws points and intervals horizontally (resp. vertically) using the xmin, x, and xmax (resp. ymin, y, and ymax) aesthetics. 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 ggplot did, and I think the tidybayes naming scheme is more intuitive: "x" and "y" are not orientations and their mapping to orientations is, in my opinion, backwards; but the base ggplot naming scheme is allowed for compatibility).

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.

normalize How to normalize heights of functions input to the thickness aesthetic. If "all" (the default), normalize so that the maximum height across all data is 1; if "panels", normalize within panels so that the maximum height in each panel is 1; if "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; if "groups", normalize within values of the opposite axis and within groups so that the maximum height in each group is 1; if "none", values are taken as is with no normalization (this should probably only be used with functions whose values are in [0,1], such as CDFs).

interval_size_domain 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 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).

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

show_slab Should the slab portion of the geom be drawn? Default TRUE.

show_point Should the point portion of the geom be drawn? Default TRUE.

show_interval Should the interval portion of the geom be drawn? Default TRUE.

dotsize          The size of the dots relative to the bin width. The default, 1, makes dots be just about as wide as the bin width.

| | |
|---|---|
| stackratio | The distance between the center of the dots in the same stack relative to the bin height. The default, 1, makes dots in the same stack just touch each other. |
| binwidth | The bin width to use for drawing the dotplots. The default value, NA, will dynamically select a bin width based on the size of the plot when drawn. |
| na.rm | If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders(). |
| geom | Use to override the default connection between stat_slabinterval and geom_slabinterval() |
| quantiles | For the stat_ and stat_dist_ stats, setting this to a value other than NA will produce a quantile dotplot: that is, a dotplot of quantiles from the sample (for stat_) or a dotplot of quantiles from the distribution (for stat_dist_). The value of quantiles determines the number of quantiles to plot. See Kay et al. (2016) and Fernandes et al. (2018) for more information on quantile dotplots. |
| point_interval | A function from the point_interval() family (e.g., median_qi, mean_qi, etc). This function should take in a vector of value, and should obey the .width and .simple_names parameters of point_interval() functions, such that when given a vector with .simple_names = TRUE should return a data frame with variables .value, .lower, .upper, and .width. Output will be converted to the appropriate x- or y-based aesthetics depending on the value of orientation. See the point_interval() family of functions for more information. |

## 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)

The stat_... and stat_dist_... versions of the stats 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).

**Value**

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

**Aesthetics**

These stats support the following aesthetics:

- x
- y
- datatype
- thickness
- size
- group

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

- x
- y
- slab_shape
- datatype
- alpha
- colour
- linetype
- fill
- shape
- stroke
- point_colour
- point_fill
- point_alpha
- point_size
- size
- interval_colour
- interval_alpha
- interval_size
- interval_linetype
- slab_size
- slab_colour
- slab_fill
- slab_alpha

- slab_linetype
- ymin
- ymax
- xmin
- xmax
- width
- height
- thickness
- group

See examples of some of these aesthetics in action in vignette("slabinterval"). Learn more about the sub-geom 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](#).

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](#).

## See Also

See [stat_sample_slabinterval()](#) and [stat_dist_slabinterval()](#) for families of other stats built on top of [geom_slabinterval()](#). See vignette("slabinterval") 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                    *Multiple uncertainty interval plots (ggplot geom)*

---

### Description

Multiple interval geoms with default aesthetics designed for use with output from [point_interval()](). Wrapper around [geom_slabinterval()]().

### Usage

```
geom_interval(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  side = "both",
  orientation = NA,
  interval_size_range = c(1, 6),
  show_slab = FALSE,
  show_point = FALSE
)
```

### Arguments

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

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

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

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

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

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

position        The position adjustment to use for overlapping points on this layer. Setting this
                equal to "dodge" can be useful if you have overlapping intervals.

...             Arguments passed on to geom_slabinterval

                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 indi-
                       cates bottom/left justification and 1 indicates top/right justification (depend-
                       ing 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.

                normalize  How to normalize heights of functions input to the thickness aes-
                       thetic. If "all" (the default), normalize so that the maximum height across
                       all data is 1; if "panels", normalize within panels so that the maximum
                       height in each panel is 1; if "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; if "groups", normalize within values of the opposite
                       axis and within groups so that the maximum height in each group is 1; if
                       "none", values are taken as is with no normalization (this should probably
                       only be used with functions whose values are in [0,1], such as CDFs).

                interval_size_domain  The minimum and maximum of the values of the size
                       aesthetic that will be translated into actual sizes for intervals drawn accord-
                       ing to interval_size_range (see the documentation for that argument.)

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

                show_interval  Should the interval portion of the geom be drawn? Default
                       TRUE.

                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 speci-
                fication, e.g. `borders()`.

side            Which side to draw the slab on. `"topright"`, `"top"`, and `"right"` are syn-
                onyms 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 this geom is drawn horizontally (`"horizontal"`) or vertically (`"vertical"`).
                The default, NA, automatically detects the orientation based on how the aesthetics
                are assigned, and should generally do an okay job at this. When horizontal (resp.
                vertical), the geom uses the y (resp. x) aesthetic to identify different groups,
                then for each group uses the x (resp. y) aesthetic and the `thickness` aesthetic
                to draw a function as an slab, and draws points and intervals horizontally (resp.
                vertically) using the xmin, x, and xmax (resp. ymin, y, and ymax) aesthetics.
                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 ggplot did, and I think the
                tidybayes naming scheme is more intuitive: `"x"` and `"y"` are not orientations
                and their mapping to orientations is, in my opinion, backwards; but the base
                ggplot naming scheme is allowed for compatibility).

interval_size_range
                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 in-
                dicates 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).

show_slab       Should the slab portion of the geom be drawn? Default TRUE.

show_point      Should the point portion of the geom be drawn? Default TRUE.

### Details

These geoms are wrappers around `geom_slabinterval()` with defaults designed to produce multi-
ple interval plots. These geoms set some default aesthetics equal to the .lower, .upper, and .width
columns generated by the `point_interval` family of functions, making them often more conve-
nient than vanilla `geom_linerange()` when used with functions like `median_qi()`, `mean_qi()`,
`mode_hdi()`, etc.

Specifically, geom_interval acts as if its default aesthetics are aes(color = fct_rev(ordered(.width))).

**Value**

A [ggplot2::Geom](#) representing a multiple interval geometry which can be added to a [ggplot()](#) object.

**Aesthetics**

These geoms support the following aesthetics:

- x
- y
- datatype
- alpha
- colour
- linetype
- fill
- shape
- stroke
- point_colour
- point_fill
- point_alpha
- point_size
- size
- interval_colour
- interval_alpha
- interval_size
- interval_linetype
- slab_size
- slab_colour
- slab_fill
- slab_alpha
- slab_linetype
- ymin
- ymax
- xmin
- xmax
- width
- height
- thickness
- group

See examples of some of these aesthetics in action in vignette("slabinterval"). Learn more about the sub-geom 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_interval()` for the stat version, intended for use on samples from a distribution. See `geom_interval()` for a similar geom intended for intervals without point summaries. See `stat_sample_slabinterval()` for a variety of other stats that combine intervals with densities and CDFs. See `geom_slabinterval()` for the geom that these geoms wrap. All parameters of that geom are available to these geoms.

## 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(.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 uncertainty 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 [aes()](#) or [aes_()](#). If specified and inherit.aes
             = TRUE (the default), it is combined with the default mapping at the top level of
             the plot. You must supply mapping if there is no plot mapping.

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

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

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

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

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

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

...          Other arguments passed to [layer()](#).

step         Should the line/ribbon be drawn as a step function? One of: FALSE (do not draw
             as a step function, the default), TRUE (draw a step function using the "mid" ap-
             proach), "mid" (draw steps midway between adjacent x values), "hv" (draw
             horizontal-then-vertical steps), "vh" (draw as vertical-then-horizontal steps).
             TRUE is an alias for "mid" because for a step function with ribbons, "mid" is
             probably what you want (for the other two step approaches the ribbons at either
             the vert first or vert last x value will not be visible).

orientation  Whether this geom is drawn horizontally ("horizontal") or vertically ("vertical").
             The default, NA, automatically detects the orientation based on how the aesthetics
             are assigned, and should generally do an okay job at this. When horizontal (resp.
             vertical), the geom uses the y (resp. x) aesthetic to identify different groups,
             then for each group uses the x (resp. y) aesthetic and the thickness aesthetic
             to draw a function as an slab, and draws points and intervals horizontally (resp.
             vertically) using the xmin, x, and xmax (resp. ymin, y, and ymax) aesthetics.
             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 ggplot did, and I think the
             tidybayes naming scheme is more intuitive: "x" and "y" are not orientations

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

### Details

geom_lineribbon is a combination version 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+uncertainty ribbon geometry which can be added to a ggplot() object.

### 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)

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 = fct_rev(ordered(.width)))
  geom_lineribbon() +
  scale_fill_brewer()
```

---

geom_pointinterval        *Point + multiple uncertainty interval plots (ggplot geom)*

---

### Description

Combined point + multiple interval geoms with default aesthetics designed for use with output from
[point_interval()](). Wrapper around [geom_slabinterval()]().

### Usage

```
geom_pointinterval(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  side = "both",
  orientation = NA,
  show_slab = FALSE,
  show.legend = c(size = FALSE)
)
```

### Arguments

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

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

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

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

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

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

position        The position adjustment to use for overlapping points on this layer. Setting this
                equal to "dodge" can be useful if you have overlapping intervals.

...             Arguments passed on to [geom_slabinterval]()

                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 indi-
cates bottom/left justification and 1 indicates top/right justification (depend-
ing 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.

normalize  How to normalize heights of functions input to the thickness aes-
thetic. If "all" (the default), normalize so that the maximum height across
all data is 1; if "panels", normalize within panels so that the maximum
height in each panel is 1; if "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; if "groups", normalize within values of the opposite
axis and within groups so that the maximum height in each group is 1; if
"none", values are taken as is with no normalization (this should probably
only be used with functions whose values are in [0,1], such as CDFs).

interval_size_domain  The minimum and maximum of the values of the size
aesthetic that will be translated into actual sizes for intervals drawn accord-
ing to interval_size_range (see the documentation for that argument.)

interval_size_range  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 in-
put 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 val-
ues (the min and max of the actual sizes used to draw intervals).

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

show_point  Should the point portion of the geom be drawn? Default TRUE.

show_interval  Should the interval portion of the geom be drawn? Default
TRUE.

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

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 specifi-
cation, e.g. [borders()]().

side  Which side to draw the slab on. "topright", "top", and "right" are syn-
onyms 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 this geom is drawn horizontally (″horizontal″) or vertically (″vertical″).
                  The default, NA, automatically detects the orientation based on how the aesthetics
                  are assigned, and should generally do an okay job at this. When horizontal (resp.
                  vertical), the geom uses the y (resp. x) aesthetic to identify different groups,
                  then for each group uses the x (resp. y) aesthetic and the thickness aesthetic
                  to draw a function as an slab, and draws points and intervals horizontally (resp.
                  vertically) using the xmin, x, and xmax (resp. ymin, y, and ymax) aesthetics.
                  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 ggplot did, and I think the
                  tidybayes naming scheme is more intuitive: ″x″ and ″y″ are not orientations
                  and their mapping to orientations is, in my opinion, backwards; but the base
                  ggplot naming scheme is allowed for compatibility).

show_slab         Should the slab portion of the geom be drawn? Default TRUE.

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).

## Details

These geoms are wrappers around geom_slabinterval() with defaults designed to produce points+interval
plots. These geoms set some default aesthetics equal to the .lower, .upper, and .width columns
generated by the point_interval family of functions, making them often more convenient than
vanilla geom_pointrange() when used with functions like median_qi(), mean_qi(), mode_hdi(),
etc.

Specifically, geom_pointinterval acts as if its default aesthetics are aes(size = -.width).

## Value

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

## Aesthetics

These geoms support the following aesthetics:

- x
- y
- datatype
- alpha
- colour
- linetype
- fill
- shape
- stroke

- `point_colour`
- `point_fill`
- `point_alpha`
- `point_size`
- `size`
- `interval_colour`
- `interval_alpha`
- `interval_size`
- `interval_linetype`
- `slab_size`
- `slab_colour`
- `slab_fill`
- `slab_alpha`
- `slab_linetype`
- `ymin`
- `ymax`
- `xmin`
- `xmax`
- `width`
- `height`
- `thickness`
- `group`

See examples of some of these aesthetics in action in `vignette("slabinterval")`. Learn more about the sub-geom 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_slabinterval()`](#) for the geom that these geoms wrap. All parameters of that geom are available to these geoms.

See [`stat_pointinterval()`](#) for the stat version, intended for use on samples from a distribution. See [`geom_interval()`](#) for a similar stat intended for intervals without point summaries. See [`stat_sample_slabinterval()`](#) for a variety of other stats that combine intervals with densities and CDFs. See [`geom_slabinterval()`](#) for the geom that these geoms wrap. All parameters of that geom are available to these geoms.

## 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_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",
  ...,
  side = c("topright", "top", "right", "bottomleft", "bottom", "left", "topleft",
    "bottomright", "both"),
  scale = 0.9,
  orientation = NA,
  justification = NULL,
  normalize = c("all", "panels", "xy", "groups", "none"),
```

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

## Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by [aes()](aes()) or [aes_()](aes_()). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |
| data | The data to be displayed in this layer. There are three options: |
| | If NULL, the default, the data is inherited from the plot data as specified in the call to [ggplot()](ggplot()). |
| | A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See [fortify()](fortify()) for which variables will be created. |
| | A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)). |
| stat | The statistical transformation to use on the data for this layer, as a string. |
| position | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| ... | Other arguments passed to [layer()](layer()). |
| side | Which side to draw 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. |
| orientation | Whether this geom is drawn horizontally ("horizontal") or vertically ("vertical"). The default, NA, automatically detects the orientation based on how the aesthetics are assigned, and should generally do an okay job at this. When horizontal (resp. vertical), the geom uses the y (resp. x) aesthetic to identify different groups, then for each group uses the x (resp. y) aesthetic and the thickness aesthetic to draw a function as an slab, and draws points and intervals horizontally (resp. vertically) using the xmin, x, and xmax (resp. ymin, y, and ymax) aesthetics. 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 ggplot did, and I think the tidybayes naming scheme is more intuitive: "x" and "y" are not orientations and their mapping to orientations is, in my opinion, backwards; but the base ggplot naming scheme is allowed for compatibility). |
| 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. |
| normalize   | How to normalize heights of functions input to the thickness aesthetic. If "all" (the default), normalize so that the maximum height across all data is 1; if "panels", normalize within panels so that the maximum height in each panel is 1; if "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; if "groups", normalize within values of the opposite axis and within groups so that the maximum height in each group is 1; if "none", values are taken as is with no normalization (this should probably only be used with functions whose values are in [0,1], such as CDFs). |

interval_size_domain

> 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

> 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).

| fatten_point | A multiplicative factor used to adjust the size of the point relative to the size of the thickest interval line. If you wish to specify point sizes directly, you can |
|--------------|-----------------------------------------------------------------------|

|              | also use the point_size aesthetic and scale_point_size_continuous() or scale_point_size_discrete(); sizes specified with that aesthetic will not be adjusted using fatten_point. |
|--------------|----------------------------------------------------------------------------|
| show_slab    | Should the slab portion of the geom be drawn? Default TRUE.                 |
| show_point   | Should the point portion of the geom be drawn? Default TRUE.                |
| show_interval| Should the interval portion of the geom be drawn? Default TRUE.             |
| na.rm        | If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed. |
| show.legend  | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. |
| inherit.aes  | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders(). |

## Details

geom_slabinterval is a flexible meta-geom that you can use directly or through a variety of "short-cut" 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 access (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 specifying 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 and stats include:

- stat_sample_slabinterval() and associated stats
- stat_dist_slabinterval() and associated stats
- geom_pointinterval() / stat_pointinterval()
- geom_interval() / stat_interval()
- geom_dots() / stat_dots()

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

These geoms support the following aesthetics:

- x
- y
- datatype
- alpha
- colour
- linetype
- fill
- shape
- stroke
- point_colour
- point_fill
- point_alpha
- point_size
- size
- interval_colour
- interval_alpha
- interval_size
- interval_linetype
- slab_size
- slab_colour
- slab_fill
- slab_alpha
- slab_linetype
- ymin
- ymax
- xmin
- xmax
- width

- height

- thickness

- group

See examples of some of these aesthetics in action in vignette("slabinterval"). Learn more
about the sub-geom 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 stat_sample_slabinterval() and stat_dist_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.
```

---

lkjcorr_marginal          *Marginal distribution of a single correlation from an LKJ distribution*

---

### Description

Marginal distribution for the correlation in a single cell from a correlation matrix distributed ac-
cording 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 length(n) > 1, the length is taken to be the number required. |

## Details

The LKJ distribution is a distribution over correlation matrices with a single parameter, $\eta$. For a given $\eta$ and a $K \times K$ correlation matrix $R$:

$$R \sim \text{LKJ}(\eta)$$

Each off-diagonal entry of $R$, $r_{ij} : i \neq j$, has the following marginal distribution (Lewandowski, Kurowicka, and Joe 2009):

$$\frac{r_{ij} + 1}{2} \sim \text{Beta}\left(\eta - 1 + \frac{K}{2}, \eta - 1 + \frac{K}{2}\right)$$

In other words, $r_{ij}$ is marginally distributed according to the above Beta distribution scaled into $(-1, 1)$.

## Value

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

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

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

## References

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.

**See Also**

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

**Examples**

```
library(dplyr)
library(ggplot2)
library(forcats)

expand.grid(
  eta = 1:6,
  K = 2:6
) %>%
  ggplot(aes(y = fct_rev(ordered(eta)), dist = "lkjcorr_marginal", arg1 = K, arg2 = eta)) +
  stat_dist_slab(p_limits = c(0,1)) +
  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 (".dist" by default) and a list column of distribution arguments (".args" by default), such as output by parse_dist(). |
| K | Dimension of the correlation matrix. Must be greater than or equal to 2. |

predicate      a bare expression for selecting the rows of data to modify. This is useful if data contains more than one row with an LKJ prior in it and you only want to modify some of the distributions; if this is the case, give row a predicate expression (such as you might supply to [dplyr::filter()](#)) that evaluates to TRUE on the rows you want to modify. If NULL (the default), all lkjcorr distributions in data are modified.

dist      The name of the column containing distribution names. See [parse_dist()](#).

args      The name of the column containing distribution arguments. See [parse_dist()](#).

### Details

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

Given a data frame representing parsed distribution specifications (such as returned by [parse_dist()](#)), this function updates any rows with .dist == "lkjcorr" so that the first argument to the distribution is equal to the specified dimension of the correlation matrix (K) and changes the distribution name to "lkjcorr_marginal", allowing the distribution to be easily visualized using the [stat_dist_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_lkjcorr(K = 2) %>%
  ggplot(aes(y = prior, dist = .dist, args = .args)) +
  stat_dist_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 [stat_dist_slabinterval()](#) and its shortcut stats (like stat_dist_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)

r_dist_name(dist_name)
```

### Arguments

| | |
|---|---|
| object | A character vector containing distribution specifications or a data frame with a column containing distribution specifications. |
| ... | Arguments passed to other implementations of parse_dist. |
| dist | The name of the output column to contain the distribution name |

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

## Details

parse_dist() can be applied to character vectors or to a data frame + bare column name of the column to parse, and returns a data frame with ″.dist″ and ″.args″ columns added. parse_dist() uses r_dist_name() to translate distribution names into names recognized by R.

r_dist_name() takes a character vector of names and translates common names into R distribution names. Names are first made into valid R names using [make.names()](), then translated (ignoring character case, ″.″, and ″_″). Thus, ″lognormal″, ″LogNormal″, ″log_normal″, ″log-Normal″, and any number of other variants all get translated into ″lnorm″.

## Value

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

- r_dist_name returns a character vector the same length as the input containing translations of the input names into distribution names R can recognize.

## See Also

See [stat_dist_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                  *Point and interval summaries for tidy data frames of draws from dis-*
                                *tributions*

---

### Description

Translates draws from distributions in a (possibly grouped) data frame into point and interval sum-
maries (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
)

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

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

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

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

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

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

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

mode_hdci(.data, ..., .width = 0.95)
```

## Arguments

| | |
|---|---|
| `.data` | Data frame (or grouped data frame as returned by [group_by()](#)) that contains draws to summarize. |
| `...` | Bare column names or expressions that, when evaluated in the context of `.data`, represent draws to summarize. If this is empty, then by default all columns that are not group columns and which are not in `.exclude` (by default `".chain"`, `".iteration"`, `".draw"`, and `".row"`) will be summarized. This can be list columns. |
| `.width` | vector of probabilities to use that determine the widths of the resulting intervals. If multiple probabilities are provided, multiple rows per group are generated, each with a different probability interval (and value of the corresponding `.width` column). |
| `.point` | Point summary function, which takes a vector and returns a single value, e.g. [mean()](#), [median()](#), or [Mode()](#). |

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

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

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

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

.prob        Deprecated. Use .width instead.

x           vector to summarize (for interval functions: qi and hdi)

## Details

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

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

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

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

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

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

hdi yields the highest-density interval(s) (also known as the highest posterior density interval). **Note:** If the distribution is multimodal, hdi may return multiple intervals for each probability level (these will be spread over rows). You may wish to use hdci (below) instead if you want a single

highest-density interval, with the caveat that when the distribution is multimodal `hdci` is not a highest-density interval. Internally hdi uses `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.

### 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))
```

---

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. `scale_point_color_discrete` passes arguments to [`scale_color_discrete()`]. See those functions for more details. |
| `aesthetics` | Names of aesthetics to set scales for. |
| `guide` | Guide to use for legends for an aesthetic. |
| `range` | a numeric vector of length 2 that specifies the minimum and maximum size of the plotting symbol after transformation. |
| `na.translate` | In discrete scales, should we show missing values? |
| `na.value` | When `na.translate` is true, what value should be shown? |
| `limits` | One of: |
| | • NULL to use the default scale range |
| | • A numeric vector of length two providing limits of the scale. Use NA to refer to the existing minimum or maximum |
| | • A function that accepts the existing (automatic) limits and returns new limits Note that setting limits on positional scales will **remove** data outside of |

the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see `coord_cartesian()`).

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

### Details

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

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

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

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

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

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

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

### Value

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

### Author(s)

Matthew Kay

### See Also

[scale_color_discrete()](scale_color_discrete()), [scale_color_continuous()](scale_color_continuous()), etc.

### 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, dist = .dist, args = .args)) +
  stat_dist_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 option 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
  )
```

---

stat_dist_slabinterval

*Distribution + interval plots (eye plots, half-eye plots, CCDF barplots, etc) for analytical distributions (ggplot stat)*

---

### Description

Stats for computing distribution functions (densities or CDFs) + intervals for use with [geom_slabinterval()](geom_slabinterval()). Uses the dist aesthetic to specify a distribution using objects from the distributional package, or using distribution names and arg1, ... arg9 aesthetics (or args as a list column) to specify distribution arguments. See *Details*.

## Usage

```
stat_dist_slabinterval(
  mapping = NULL,
  data = NULL,
  geom = "slabinterval",
  position = "identity",
  ...,
  slab_type = c("pdf", "cdf", "ccdf"),
  p_limits = c(0.001, 0.999),
  orientation = NA,
  limits = NULL,
  n = 501,
  .width = c(0.66, 0.95),
  show_slab = TRUE,
  show_interval = TRUE,
  na.rm = FALSE,
  show.legend = c(size = FALSE),
  inherit.aes = TRUE
)

stat_dist_halfeye(...)

stat_dist_eye(..., side = "both")

stat_dist_ccdfinterval(
  ...,
  slab_type = "ccdf",
  justification = 0.5,
  side = "topleft",
  normalize = "none"
)

stat_dist_cdfinterval(
  ...,
  slab_type = "cdf",
  justification = 0.5,
  side = "topleft",
  normalize = "none"
)

stat_dist_gradientinterval(
  mapping = NULL,
  data = NULL,
  geom = "slabinterval",
  position = "identity",
  ...,
  justification = 0.5,
  thickness = 1,
```

```
    show.legend = c(size = FALSE, slab_alpha = FALSE),
    inherit.aes = TRUE
)

stat_dist_pointinterval(..., show_slab = FALSE)

stat_dist_interval(
    mapping = NULL,
    data = NULL,
    geom = "interval",
    position = "identity",
    ...,
    show_slab = FALSE,
    show_point = FALSE,
    show.legend = NA,
    inherit.aes = TRUE
)

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

## Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by [aes()](#) or [aes_()](#). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |
| data | The data to be displayed in this layer. There are three options: |
| | If NULL, the default, the data is inherited from the plot data as specified in the call to [ggplot()](#). |
| | A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See [fortify()](#) for which variables will be created. |
| | A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)). |
| geom | Use to override the default connection between stat_slabinterval and [geom_slabinterval()](#) |
| position | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| ... | Other arguments passed to [layer()](#). They may also be arguments to the paired geom (e.g., [geom_pointinterval()](#)) |

| | |
|---|---|
| slab_type | The type of slab function to calculate: probability density (or mass) function (″pdf″), cumulative distribution function (″cdf″), or complementary CDF (″ccdf″). |
| p_limits | Probability limits (as a vector of size 2) used to determine the lower and upper limits of the slab. E.g., if this is c(.001,.999) (the default), then a slab is drawn for the distribution from the quantile at p = .001 to the quantile at p = .999. |
| orientation | Whether this geom is drawn horizontally (″horizontal″) or vertically (″vertical″). The default, NA, automatically detects the orientation based on how the aesthetics are assigned, and should generally do an okay job at this. When horizontal (resp. vertical), the geom uses the y (resp. x) aesthetic to identify different groups, then for each group uses the x (resp. y) aesthetic and the thickness aesthetic to draw a function as an slab, and draws points and intervals horizontally (resp. vertically) using the xmin, x, and xmax (resp. ymin, y, and ymax) aesthetics. 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 ggplot did, and I think the tidybayes naming scheme is more intuitive: ″x″ and ″y″ are not orientations and their mapping to orientations is, in my opinion, backwards; but the base ggplot naming scheme is allowed for compatibility). |
| 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 slab_function |
| .width | The .width argument passed to interval_function or point_interval. |
| show_slab | Should the slab portion of the geom be drawn? Default TRUE. |
| show_interval | Should the interval portion of the geom be drawn? Default TRUE. |
| 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()](). |
| side | Which side to draw 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). |

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.

normalize       How to normalize heights of functions input to the thickness aesthetic. If
                "all" (the default), normalize so that the maximum height across all data is
                1; if "panels", normalize within panels so that the maximum height in each
                panel is 1; if "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;
                if "groups", normalize within values of the opposite axis and within groups so
                that the maximum height in each group is 1; if "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).

thickness       Override for the thickness aesthetic in [geom_slabinterval()](): the thickness
                of the slab at each x / y value of the slab (depending on orientation).

show_point      Should the point portion of the geom be drawn? Default TRUE.

## Details

A highly configurable stat for generating a variety of plots that combine a "slab" that describes a
distribution plus an interval. Several "shortcut" stats are provided which combine multiple options
to create useful geoms, particularly *eye plots* (a combination of a violin plot and interval), *half-eye
plots* (a density plus interval), and *CCDF bar plots* (a complementary CDF plus interval).

The shortcut stat names follow the pattern stat_dist_[name].

Stats include:

- stat_dist_eye: Eye plots (violin + interval)

- stat_dist_halfeye: Half-eye plots (density + interval)

- stat_dist_ccdfinterval: CCDF bar plots (CCDF + interval)

- stat_dist_cdfinterval: CDF bar plots (CDF + interval)

- stat_dist_gradientinterval: Density gradient + interval plots

- stat_dist_pointinterval: Point + interval plots

- stat_dist_interval: Interval plots

These stats expect a dist aesthetic to specify a distribution. This aesthetic can be used in one of
two ways:

- dist can be any distribution object from the [distributional]() package, such as [dist_normal()](),
  [dist_beta()](), etc. 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 aes-
  thetics (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.

### Aesthetics

These stats support the following aesthetics:

- dist
- args
- arg1
- arg2
- arg3
- arg4
- arg5
- arg6
- arg7
- arg8
- arg9
- x
- y
- datatype
- thickness
- size
- group

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

- x
- y
- datatype
- alpha
- colour

- linetype
- fill
- shape
- stroke
- point_colour
- point_fill
- point_alpha
- point_size
- size
- interval_colour
- interval_alpha
- interval_size
- interval_linetype
- slab_size
- slab_colour
- slab_fill
- slab_alpha
- slab_linetype
- ymin
- ymax
- xmin
- xmax
- width
- height
- thickness
- group

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

#### Computed Variables

- 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.

- 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.

**See Also**

See geom_slabinterval() for more information on the geom these stats use by default and some of the options they have. See stat_sample_slabinterval() for the versions of these stats that can be used on samples. See vignette("slabinterval") for a variety of examples of use.

**Examples**

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

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

dist_df %>%
  ggplot(aes(x = group, dist = "norm", arg1 = mean, arg2 = sd, fill = subgroup)) +
  stat_dist_eye(position = "dodge")

# 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, dist = dist_normal(mean, sd), fill = subgroup)) +
  stat_dist_eye(position = "dodge")

# the stat_dist_... family applies a Jacobian adjustment to densities
# when plotting on transformed scales in order to plot them correctly.
# For example, here is a log-Normal distribution plotted on the log
# scale, where it will appear Normal:
data.frame(dist = "lnorm") %>%
  ggplot(aes(y = 1, dist = dist, arg1 = log(10), arg2 = 2*log(10))) +
  stat_dist_halfeye() +
  scale_x_log10(breaks = 10^seq(-5,7, by = 2))

# see vignette("slabinterval") for many more examples.
```

---

stat_interval                  *Multiple uncertainty interval plots (ggplot stat)*

---

**Description**

A combination of [stat_sample_slabinterval()](#) and [geom_slabinterval()](#) with sensible de-
faults. While the corresponding geoms are intended for use on data frames that have already been
summarized using a [point_interval()](#) function, these `stats` are intended for use directly on data
frames of draws, and will perform the summarization using a [point_interval()](#) function.

**Usage**

```
stat_interval(
  mapping = NULL,
  data = NULL,
  geom = "interval",
  position = "identity",
  ...,
  orientation = NA,
  interval_function = NULL,
  interval_args = list(),
  point_interval = median_qi,
  .width = c(0.5, 0.8, 0.95),
  show_point = FALSE,
  show_slab = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  .prob,
  fun.data,
  fun.args
)
```

**Arguments**

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

| | |
|---|---|
| ... | Other arguments passed to [layer()](). They may also be arguments to the paired geom (e.g., [geom_pointinterval()]()) |
| orientation | Whether this geom is drawn horizontally ("horizontal") or vertically ("vertical"). The default, NA, automatically detects the orientation based on how the aesthetics are assigned, and should generally do an okay job at this. When horizontal (resp. vertical), the geom uses the y (resp. x) aesthetic to identify different groups, then for each group uses the x (resp. y) aesthetic and the thickness aesthetic to draw a function as an slab, and draws points and intervals horizontally (resp. vertically) using the xmin, x, and xmax (resp. ymin, y, and ymax) aesthetics. 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 ggplot did, and I think the tidybayes naming scheme is more intuitive: "x" and "y" are not orientations and their mapping to orientations is, in my opinion, backwards; but the base ggplot naming scheme is allowed for compatibility). |
| interval_function | |
| | Custom function for generating intervals (for most common use cases the point_interval argument will be easier to use). This function takes a data frame of aesthetics and a .width parameter (a vector of interval widths), and returns a data frame with columns .width (from the .width vector), .value (point summary) and .lower and .upper (endpoints of the intervals, given the .width). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of orientation. If interval_function is NULL, point_interval is used instead. |
| interval_args | Additional arguments passed to interval_function or point_interval. |
| point_interval | A function from the [point_interval()]() family (e.g., median_qi, mean_qi, etc). This function should take in a vector of value, and should obey the .width and .simple_names parameters of [point_interval()]() functions, such that when given a vector with .simple_names = TRUE should return a data frame with variables .value, .lower, .upper, and .width. Output will be converted to the appropriate x- or y-based aesthetics depending on the value of orientation. See the [point_interval()]() family of functions for more information. |
| .width | The .width argument passed to interval_function or point_interval. |
| show_point | Should the point portion of the geom be drawn? Default TRUE. |
| show_slab | Should the slab portion of the geom be drawn? Default TRUE. |
| 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()](). |
| .prob | Deprecated. Use .width instead. |

| fun.data | Deprecated. Use `point_interval` or `interval_function` instead. |
|---|---|
| fun.args | Deprecated. Use `interval_args` instead. |

### Value

A [ggplot2::Stat](#) representing a multiple interval geometry which can be added to a [ggplot()](#) object.

### Aesthetics

These stats support the following aesthetics:

- `colour`
- `datatype`
- `x`
- `y`
- `thickness`
- `group`

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

- `x`
- `y`
- `datatype`
- `alpha`
- `colour`
- `linetype`
- `fill`
- `shape`
- `stroke`
- `point_colour`
- `point_fill`
- `point_alpha`
- `point_size`
- `size`
- `interval_colour`
- `interval_alpha`
- `interval_size`
- `interval_linetype`
- `slab_size`
- `slab_colour`
- `slab_fill`

- slab_alpha

- slab_linetype

- ymin

- ymax

- xmin

- xmax

- width

- height

- thickness

- group

See examples of some of these aesthetics in action in vignette("slabinterval"). Learn more about the sub-geom 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 versions, intended for use on points and intervals that have already been summarized using a point_interval() function. See stat_pointinterval() for a similar stat intended for point summaries and intervals. See stat_sample_slabinterval() for a variety of other stats that combine intervals with densities and CDFs. See geom_slabinterval() for the geom that these geoms wrap. All parameters of that geom are available to these geoms.

### Examples

```
library(dplyr)
library(ggplot2)

data(RankCorr_u_tau, package = "ggdist")

RankCorr_u_tau %>%
  group_by(i) %>%
  ggplot(aes(y = factor(i), x = u_tau)) +
  stat_interval() +
  scale_color_brewer()

RankCorr_u_tau %>%
  group_by(i) %>%
  ggplot(aes(x = factor(i), y = u_tau)) +
  stat_interval() +
  scale_color_brewer()
```

---

stat_lineribbon                *Line + multiple probability ribbon plots (ggplot stat)*

---

### Description

A combination of [stat_slabinterval()](#) and [geom_lineribbon()](#) with sensible defaults. 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, and will perform the summarization using a [point_interval()](#) function; stat_dist_lineribbon
is intended for use on analytical distributions through the dist, arg1, ... arg9, and args aesthetics.

### Usage

```
stat_lineribbon(
  mapping = NULL,
  data = NULL,
  geom = "lineribbon",
  position = "identity",
  ...,
  interval_function = NULL,
  interval_args = list(),
  point_interval = median_qi,
  .width = c(0.5, 0.8, 0.95),
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  .prob,
  fun.data,
  fun.args
)

stat_dist_lineribbon(
  mapping = NULL,
  data = NULL,
  geom = "lineribbon",
  position = "identity",
  ...,
  n = 501,
  .width = c(0.5, 0.8, 0.95),
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

### Arguments

mapping            Set of aesthetic mappings created by [aes()](#) or [aes_()](#). If specified and inherit.aes
                   = TRUE (the default), it is combined with the default mapping at the top level of

|  | the plot. You must supply mapping if there is no plot mapping. |
| --- | --- |
| data | The data to be displayed in this layer. There are three options: |
|  | If NULL, the default, the data is inherited from the plot data as specified in the call to [ggplot()](). |
|  | A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See [fortify()]() for which variables will be created. |
|  | A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)). |
| geom | Use to override the default connection between geom_lineribbon and stat_lineribbon. |
| position | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| ... | Other arguments passed to [layer().]() They may also be arguments to the paired geom (e.g., [geom_pointinterval()]()) |
| interval_function | |
|  | Custom function for generating intervals (for most common use cases the point_interval argument will be easier to use). This function takes a data frame of aesthetics and a .width parameter (a vector of interval widths), and returns a data frame with columns .width (from the .width vector), .value (point summary) and .lower and .upper (endpoints of the intervals, given the .width). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of orientation. If interval_function is NULL, point_interval is used instead. |
| interval_args | Additional arguments passed to interval_function or point_interval. |
| point_interval | A function from the [point_interval()]() family (e.g., median_qi, mean_qi, etc). This function should take in a vector of value, and should obey the .width and .simple_names parameters of [point_interval()]() functions, such that when given a vector with .simple_names = TRUE should return a data frame with variables .value, .lower, .upper, and .width. Output will be converted to the appropriate x- or y-based aesthetics depending on the value of orientation. See the [point_interval()]() family of functions for more information. |
| .width | The .width argument passed to interval_function or point_interval. |
| na.rm | If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed. |
| show.legend | Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. [borders()](). |
| .prob | Deprecated. Use .width instead. |
| fun.data | Deprecated. Use point_interval or interval_function instead. |
| fun.args | Deprecated. Use interval_args instead. |
| n | Number of points at which to evaluate slab_function |

**Value**

A [ggplot2::Stat](#) representing a combined line+uncertainty ribbon geometry which can be added to
a [ggplot()](#) object.

**See Also**

See [geom_lineribbon()](#) for the geom version, intended for use on points and intervals that have
already been summarized using a [point_interval()](#) function. See [stat_pointinterval()](#) for a
similar stat intended for point summaries and intervals.

**Examples**

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

tibble(x = 1:10) %>%
  group_by_all() %>%
  do(tibble(y = rnorm(100, .$x))) %>%
  ggplot(aes(x = x, y = y)) +
  stat_lineribbon() +
  scale_fill_brewer()

tibble(
  x = 1:10,
  sd = seq(1, 3, length.out = 10)
) %>%
  ggplot(aes(x = x, dist = dist_normal(x, sd))) +
  stat_dist_lineribbon() +
  scale_fill_brewer()
```

---

stat_pointinterval          *Point summary + multiple uncertainty interval plots (ggplot stat)*

---

**Description**

A combination of [stat_sample_slabinterval()](#) and [geom_slabinterval()](#) with sensible de-
faults. While the corresponding geoms are intended for use on data frames that have already been
summarized using a [point_interval()](#) function, these stats are intended for use directly on data
frames of draws, and will perform the summarization using a [point_interval()](#) function.

**Usage**

```
stat_pointinterval(
  mapping = NULL,
  data = NULL,
```

```
  geom = "pointinterval",
  position = "identity",
  ...,
  orientation = NA,
  interval_function = NULL,
  interval_args = list(),
  point_interval = median_qi,
  .width = c(0.66, 0.95),
  show_slab = FALSE,
  na.rm = FALSE,
  show.legend = c(size = FALSE),
  inherit.aes = TRUE,
  .prob,
  fun.data,
  fun.args
)
```

**Arguments**

| | |
|---|---|
| mapping | Set of aesthetic mappings created by [aes()](#) or [aes_()](#). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |
| data | The data to be displayed in this layer. There are three options: |
| | If NULL, the default, the data is inherited from the plot data as specified in the call to [ggplot()](#). |
| | A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See [fortify()](#) for which variables will be created. |
| | A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)). |
| geom | Use to override the default connection between stat_slabinterval and [geom_slabinterval()](#) |
| position | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| ... | Other arguments passed to [layer()](#). They may also be arguments to the paired geom (e.g., [geom_pointinterval()](#)) |
| orientation | Whether this geom is drawn horizontally ("horizontal") or vertically ("vertical"). The default, NA, automatically detects the orientation based on how the aesthetics are assigned, and should generally do an okay job at this. When horizontal (resp. vertical), the geom uses the y (resp. x) aesthetic to identify different groups, then for each group uses the x (resp. y) aesthetic and the thickness aesthetic to draw a function as an slab, and draws points and intervals horizontally (resp. vertically) using the xmin, x, and xmax (resp. ymin, y, and ymax) aesthetics. 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 ggplot did, and I think the tidybayes naming scheme is more intuitive: "x" and "y" are not orientations |

and their mapping to orientations is, in my opinion, backwards; but the base ggplot naming scheme is allowed for compatibility).

interval_function
Custom function for generating intervals (for most common use cases the point_interval argument will be easier to use). This function takes a data frame of aesthetics and a .width parameter (a vector of interval widths), and returns a data frame with columns .width (from the .width vector), .value (point summary) and .lower and .upper (endpoints of the intervals, given the .width). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of orientation. If interval_function is NULL, point_interval is used instead.

interval_args     Additional arguments passed to interval_function or point_interval.

point_interval    A function from the [point_interval()](#) family (e.g., median_qi, mean_qi, etc). This function should take in a vector of value, and should obey the .width and .simple_names parameters of [point_interval()](#) functions, such that when given a vector with .simple_names = TRUE should return a data frame with variables .value, .lower, .upper, and .width. Output will be converted to the appropriate x- or y-based aesthetics depending on the value of orientation. See the [point_interval()](#) family of functions for more information.

.width            The .width argument passed to interval_function or point_interval.

show_slab         Should the slab portion of the geom be drawn? Default TRUE.

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()](#).

.prob             Deprecated. Use .width instead.

fun.data          Deprecated. Use point_interval or interval_function instead.

fun.args          Deprecated. Use interval_args instead.

## Value

A [ggplot2::Stat](#) representing a point+multiple uncertainty interval geometry which can be added to a [ggplot()](#) object.

## Aesthetics

These stats support the following aesthetics:

- datatype

- x

- y
- thickness
- size
- group

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

- x
- y
- datatype
- alpha
- colour
- linetype
- fill
- shape
- stroke
- point_colour
- point_fill
- point_alpha
- point_size
- size
- interval_colour
- interval_alpha
- interval_size
- interval_linetype
- slab_size
- slab_colour
- slab_fill
- slab_alpha
- slab_linetype
- ymin
- ymax
- xmin
- xmax
- width
- height
- thickness
- group

See examples of some of these aesthetics in action in `vignette("slabinterval")`. Learn more about the sub-geom 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 versions, intended for use on points and intervals that have already been summarized using a `point_interval()` function. See `stat_interval()` for a similar stat intended for intervals without point summaries. See `stat_sample_slabinterval()` for a variety of other stats that combine intervals with densities and CDFs.

See `geom_pointinterval()` for the geom versions, intended for use on points and intervals that have already been summarized using a `point_interval()` function. See `stat_interval()` for a similar stat intended for intervals without point summaries. See `stat_sample_slabinterval()` for a variety of other stats that combine intervals with densities and CDFs. See `geom_slabinterval()` for the geom that these geoms wrap. All parameters of that geom are available to these geoms.

**Examples**

```
library(dplyr)
library(ggplot2)

data(RankCorr_u_tau, package = "ggdist")

RankCorr_u_tau %>%
  ggplot(aes(y = factor(i), x = u_tau)) +
  stat_pointinterval(.width = c(.66, .95))

RankCorr_u_tau %>%
  ggplot(aes(x = factor(i), y = u_tau)) +
  stat_pointinterval(.width = c(.66, .95))
```

---

stat_sample_slabinterval

*Distribution + interval plots (eye plots, half-eye plots, CCDF barplots, etc) for samples (ggplot stat)*

---

**Description**

Stats for computing densities and CDFs + intervals from samples for use with `geom_slabinterval()`. Useful for creating eye plots, half-eye plots, CCDF bar plots etc.

**Usage**

```
stat_sample_slabinterval(
  mapping = NULL,
  data = NULL,
  geom = "slabinterval",
  position = "identity",
  ...,
  slab_type = c("pdf", "cdf", "ccdf", "histogram"),
```

```
      adjust = 1,
      trim = TRUE,
      breaks = "Sturges",
      outline_bars = FALSE,
      orientation = NA,
      limits = NULL,
      n = 501,
      interval_function = NULL,
      interval_args = list(),
      point_interval = median_qi,
      .width = c(0.66, 0.95),
      na.rm = FALSE,
      show.legend = c(size = FALSE),
      inherit.aes = TRUE
    )

    stat_halfeye(...)

    stat_eye(..., side = "both")

    stat_ccdfinterval(
      ...,
      slab_type = "ccdf",
      justification = 0.5,
      side = "topleft",
      normalize = "none"
    )

    stat_cdfinterval(
      ...,
      slab_type = "cdf",
      justification = 0.5,
      side = "topleft",
      normalize = "none"
    )

    stat_gradientinterval(
      mapping = NULL,
      data = NULL,
      geom = "slabinterval",
      position = "identity",
      ...,
      justification = 0.5,
      thickness = 1,
      show.legend = c(size = FALSE, slab_alpha = FALSE),
      inherit.aes = TRUE
    )
```

```
stat_histinterval(..., slab_type = "histogram")

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

## Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by [aes()](#) or [aes_()](#). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |
| data | The data to be displayed in this layer. There are three options:<br><br>If NULL, the default, the data is inherited from the plot data as specified in the call to [ggplot()](#).<br><br>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.<br><br>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_slabinterval and [geom_slabinterval()](#) |
| position | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| ... | Other arguments passed to [layer()](#). They may also be arguments to the paired geom (e.g., [geom_pointinterval()](#)) |
| slab_type | The type of slab function to calculate: probability density (or mass) function ("pdf"), cumulative distribution function ("cdf"), complementary CDF ("ccdf"), or histogram ("histogram". |
| adjust | If slab_type is "pdf", bandwidth for the density estimator is adjusted by multiplying it by this value. See [density()](#) for more information. |
| trim | If slab_type is "pdf", should the density estimate be trimmed to the range of the input data? Default TRUE. |
| breaks | If slab_type is "histogram", the breaks parameter that is passed to [hist()](#) to determine where to put breaks in the histogram. |
| outline_bars | If slab_type is "histogram", outline_bars 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. |

orientation
: Whether this geom is drawn horizontally ("horizontal") or vertically ("vertical"). The default, NA, automatically detects the orientation based on how the aesthetics are assigned, and should generally do an okay job at this. When horizontal (resp. vertical), the geom uses the y (resp. x) aesthetic to identify different groups, then for each group uses the x (resp. y) aesthetic and the thickness aesthetic to draw a function as an slab, and draws points and intervals horizontally (resp. vertically) using the xmin, x, and xmax (resp. ymin, y, and ymax) aesthetics. 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 ggplot did, and I think the tidybayes naming scheme is more intuitive: "x" and "y" are not orientations and their mapping to orientations is, in my opinion, backwards; but the base ggplot naming scheme is allowed for compatibility).

limits
: Limits for slab_function, as a vector of length two. These limits are combined with those computed by the limits_function 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.

n
: Number of points at which to evaluate slab_function

interval_function
: Custom function for generating intervals (for most common use cases the point_interval argument will be easier to use). This function takes a data frame of aesthetics and a .width parameter (a vector of interval widths), and returns a data frame with columns .width (from the .width vector), .value (point summary) and .lower and .upper (endpoints of the intervals, given the .width). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of orientation. If interval_function is NULL, point_interval is used instead.

interval_args
: Additional arguments passed to interval_function or point_interval.

point_interval
: A function from the [point_interval()](#) family (e.g., median_qi, mean_qi, etc). This function should take in a vector of value, and should obey the .width and .simple_names parameters of [point_interval()](#) functions, such that when given a vector with .simple_names = TRUE should return a data frame with variables .value, .lower, .upper, and .width. Output will be converted to the appropriate x- or y-based aesthetics depending on the value of orientation. See the [point_interval()](#) family of functions for more information.

.width
: The .width argument passed to interval_function or point_interval.

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()](#).

side                  Which side to draw the slab on. "topright", "top", and "right" are syn-
                      onyms 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).

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.

normalize             How to normalize heights of functions input to the thickness aesthetic. If
                      "all" (the default), normalize so that the maximum height across all data is
                      1; if "panels", normalize within panels so that the maximum height in each
                      panel is 1; if "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;
                      if "groups", normalize within values of the opposite axis and within groups so
                      that the maximum height in each group is 1; if "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).

thickness             Override for the thickness aesthetic in [geom_slabinterval()](): the thickness
                      of the slab at each x / y value of the slab (depending on orientation).

### Details

A highly configurable stat for generating a variety of plots that combine a "slab" that summarizes
a sample plus an interval. Several "shortcut" stats are provided which combine multiple options to
create useful geoms, particularly *eye* plots (a combination of a violin plot and interval), *half-eye*
plots (a density plus interval), and *CCDF bar plots* (a complementary CDF plus interval). These
can be handy for visualizing posterior distributions in Bayesian inference, amongst other things.

The shortcut stat names follow the pattern stat_[name].

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_histinterval: Histogram + interval plots
- stat_pointinterval: Point + interval plots
- stat_interval: Interval plots

**Value**

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

**Aesthetics**

These stats support the following aesthetics:

- x
- y
- datatype
- thickness
- size
- group

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

- x
- y
- datatype
- alpha
- colour
- linetype
- fill
- shape
- stroke
- point_colour
- point_fill
- point_alpha
- point_size
- size
- interval_colour
- interval_alpha
- interval_size
- interval_linetype
- slab_size
- slab_colour
- slab_fill
- slab_alpha
- slab_linetype

- ymin

- ymax

- xmin

- xmax

- width

- height

- thickness

- group

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

### Computed Variables

- 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.

- f: For slabs, the output values from the slab function.

- n: For slabs, the number of data points summarized into that slab.

### See Also

See geom_slabinterval() for more information on the geom these stats use by default and some of the options they have. See stat_dist_slabinterval() for the versions of these stats that can be used on analytical distributions. See vignette("slabinterval") for a variety of examples of use.

### Examples

```
library(dplyr)
library(tidyr)
library(ggplot2)

# consider the following example data:
set.seed(1234)
df = tribble(
  ~group, ~subgroup, ~value,
  "a",          "h", rnorm(500, mean = 5),
  "b",          "h", rnorm(500, mean = 7, sd = 1.5),
  "c",          "h", rnorm(500, mean = 8),
  "c",          "i", rnorm(500, mean = 9),
  "c",          "j", rnorm(500, mean = 7)
) %>%
  unnest(value)
```

```
# 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 generated by a stat_dist_... function).
# 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(f*n)))

# 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 = 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 `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_dist_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_dist_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_left_horizontal()

facet_title_right_horizontal()

axis_titles_bottom_left()
```

### 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