Package: ggdist (via r-universe)

October 25, 2024

Title Visualizations of Distributions and Uncertainty

Version 3.3.2.9000

Date 2024-03-03

Maintainer Matthew Kay <mjskay@northwestern.edu>

Description Provides primitives for visualizing distributions using 'ggplot2' that are particularly tuned for visualizing uncertainty in either a frequentist or Bayesian mode. Both analytical distributions (such as frequentist confidence distributions or Bayesian priors) and distributions represented as samples (such as bootstrap distributions or Bayesian posterior samples) are easily visualized. Visualization primitives include but are not limited to: points with multiple uncertainty intervals, eye plots (Spiegelhalter D., 1999) <https://ideas.repec.org/a/bla/jorssa/v162y1999i1p45-58.html>, density plots, gradient plots, dot plots (Wilkinson L., 1999) <doi:10.1080/00031305.1999.10474474>, 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 (>= 4.0.0)

Imports grid, ggplot2 (>= 3.5.0), scales, rlang (>= 0.3.0), cli, tibble, vctrs, withr, glue, gtable, distributional (>= 0.3.2), numDeriv, quadprog, Rcpp

Suggests tidyselect, dplyr (>= 1.0.0), fda, posterior (>= 1.4.0), beeswarm (>= 0.4.0), rmarkdown, knitr, testthat (>= 3.0.0), vdiffr (>= 1.0.0), svglite (>= 2.1.0), fontquiver, sysfonts, showtext, mvtnorm, covr, broom (>= 0.5.6), patchwork, tidyr (>= 1.0.0), ragg (>= 1.3.0), pkgdown

License GPL (>= 3)

Language en-US

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

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

VignetteBuilder knitr

RoxygenNote 7.3.2

LazyData true

Encoding UTF-8

Collate ``ggdist-package.R'' ``util.R'' ``compat.R'' ``rd.R'' ``RcppExports.R''
``abstract_geom.R'' ``abstract_stat.R''
``abstract_stat_slabinterval.R'' ``auto_partial.R''
``binning_methods.R'' ``bounder.R'' ``curve_interval.R''
``cut_cdf_qi.R'' ``data.R'' ``density.R'' ``distributions.R''
``draw_key_slabinterval.R'' ``geom.R'' ``geom_slabinterval.R''
``geom_dotsinterval.R'' ``geom_blur_dots.R'' ``geom_interval.R''
``geom_lineribbon.R'' ``geom_pointinterval.R'' ``geom_slab.R''
``geom_spike.R'' ``geom_swarm.R'' ``guide_rampbar.R''
``lkjcorr_marginal.R'' ``parse_dist.R'' ``partial_colour_ramp.R''
``point_interval.R'' ``position_dodgejust.R'' ``pr.R'' ``rd_density.R''
``rd_dotsinterval.R'' ``rd_slabinterval.R'' ``rd_spike.R''
``rd_lineribbon.R'' ``scale_colour_ramp.R'' ``scale_thickness.R''
``scale_side_mirrored.R'' ``scale_.R'' ``smooth.R'' ``stat.R''
``stat_slabinterval.R'' ``stat_dotsinterval.R'' ``stat_mcse_dots.R''
``stat_pointinterval.R'' ``stat_interval.R'' ``stat_lineribbon.R''
``stat_spike.R'' ``student_t.R'' ``subguide.R'' ``subscale.R''
``testthat.R'' ``theme_ggdist.R'' ``thickness.R''
``tidy_format_translators.R'' ``weighted_ecdf.R'' ``weighted_hist.R''
``weighted_quantile.R'' ``deprecated.R''

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

LinkingTo Rcpp

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

RemoteUrl https://github.com/mjskay/ggdist

RemoteRef HEAD

RemoteSha 5b771834afefc29609ec770ad8e9b6bf529960bf

Contents

ggdist-package 4
align 5
auto_partial 7
bandwidth 9
bin_dots 10
blur 12
bounder_cdf 13
bounder_cooke 15

bounder_range 16
breaks 16
curve_interval 19
cut_cdf_qi 23
density_bounded 24
density_histogram 28
density_unbounded 30
find_dotplot_binwidth 33
geom_blur_dots 34
geom_dots 43
geom_dotsinterval 51
geom_interval 61
geom_lineribbon 66
geom_pointinterval 70
geom_slab 75
geom_slabinterval 80
geom_spike 88
geom_swarm 93
geom_weave 101
ggdist-deprecated 110
guide_rampbar 111
lkjcorr_marginal 113
marginalize_lkjcorr 115
parse_dist 117
partial_colour_ramp 120
point_interval 121
position_dodgejust 127
Pr_ 129
ramp_colours 131
scale_colour_ramp 132
scale_side_mirrored 134
scale_thickness 137
smooth_density 141
smooth_discrete 143
smooth_none 145
stat_ccdfinterval 146
stat_cdfinterval 156
stat_dots 167
stat_dotsinterval 176
stat_eye 187
stat_gradientinterval 197
stat_halfeye 208
stat_histinterval 218
stat_interval 229
stat_lineribbon 235
stat_mcse_dots 240
stat_pointinterval 249
stat_ribbon 256

stat_slab . . . 261
stat_slabinterval . . . 270
stat_spike . . . 281
student_t . . . 289
sub-geometry-scales . . . 291
subguide_axis . . . 296
subguide_none . . . 299
subscale_identity . . . 299
subscale_thickness . . . 300
theme_ggdist . . . 302
thickness . . . 303
tidy-format-translators . . . 304
waiver . . . 306
weighted_ecdf . . . 307
weighted_quantile . . . 308

Index **311**

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

Description

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

Details

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

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

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

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

Author(s)

Maintainer: Matthew Kay <mjskay@northwestern.edu>

Other contributors:

- Brenton M. Wiernik <brenton@wiernik.org> [contributor]

See Also

Useful links:

- https://mjskay.github.io/ggdist/
- https://github.com/mjskay/ggdist/
- Report bugs at https://github.com/mjskay/ggdist/issues/new

---

align — *Break (bin) alignment methods*

Description

Methods for aligning breaks (bins) in histograms, as used in the `align` argument to `density_histogram()`.

Supports automatic partial function application with waived arguments.

Usage

```
align_none(breaks)

align_boundary(breaks, at = 0)

align_center(breaks, at = 0)
```

Arguments

`breaks`	<numeric> A sorted vector of breaks (bin edges).
`at`	<scalar numeric> The alignment point. • For `align_boundary()`: align breaks so that a bin edge lines up with `at`. • For `align_center()`: align breaks so that the center of a bin lines up with `at`.

Details

These functions take a sorted vector of equally-spaced `breaks` giving bin edges and return a numeric offset which, if subtracted from `breaks`, will align them as desired:

- `align_none()` performs no alignment (it always returns `0`).
- `align_boundary()` ensures that a bin edge lines up with `at`.
- `align_center()` ensures that a bin center lines up with `at`.

For `align_boundary()` (respectively `align_center()`), if no bin edge (or center) in the range of `breaks` would line up with `at`, it ensures that `at` is an integer multiple of the bin width away from a bin edge (or center).

Value

A scalar numeric returning an offset to be subtracted from `breaks`.

See Also

`density_histogram()`, breaks

Examples

```
library(ggplot2)

set.seed(1234)
x = rnorm(200, 1, 2)

# If we manually specify a bin width using breaks_fixed(), the default
# alignment (align_none()) will not align bin edges to any "pretty" numbers.
# Here is a comparison of the three alignment methods on such a histogram:
ggplot(data.frame(x), aes(x)) +
  stat_slab(
    aes(y = "align_none()\nor 'none'"),
    density = "histogram",
    breaks = breaks_fixed(width = 1),
    outline_bars = TRUE,
    # no need to specify align; align_none() is the default
    color = "black",
  ) +
  stat_slab(
    aes(y = "align_center(at = 0)\nor 'center'"),
    density = "histogram",
    breaks = breaks_fixed(width = 1),
    align = align_center(at = 0),   # or align = "center"
    outline_bars = TRUE,
    color = "black",
  ) +
  stat_slab(
    aes(y = "align_boundary(at = 0)\nor 'boundary'"),
    density = "histogram",
    breaks = breaks_fixed(width = 1),
    align = align_boundary(at = 0), # or align = "boundary"
    outline_bars = TRUE,
    color = "black",
  ) +
  geom_point(aes(y = 0.7), alpha = 0.5) +
  labs(
    subtitle = "ggdist::stat_slab(density = 'histogram', ...)",
    y = "align =",
    x = NULL
  ) +
  geom_vline(xintercept = 0, linetype = "22", color = "red")
```

auto_partial *Automatic partial function application in ggdist*

Description

Several **ggdist** functions support *automatic partial application*: when called, if all of their required arguments have not been provided, the function returns a modified version of itself that uses the arguments passed to it so far as defaults. Technically speaking, these functions are essentially "Curried" with respect to their required arguments, but I think "automatic partial application" gets the idea across more clearly.

Functions supporting automatic partial application include:

- The `point_interval()` family, such as `median_qi()`, `mean_qi()`, `mode_hdi()`, etc.
- The `smooth_` family, such as `smooth_bounded()`, `smooth_unbounded()`, `smooth_discrete()`, and `smooth_bar()`.
- The `density_` family, such as `density_bounded()`, `density_unbounded()` and `density_histogram()`.
- The align family.
- The breaks family.
- The bandwidth family.
- The blur family.

Partial application makes it easier to supply custom parameters to these functions when using them inside other functions, such as geoms and stats. For example, smoothers for `geom_dots()` can be supplied in one of three ways:

- as a suffix: `geom_dots(smooth = "bounded")`
- as a function: `geom_dots(smooth = smooth_bounded)`
- as a partially-applied function with options: `geom_dots(smooth = smooth_bounded(kernel = "cosine"))`

Many other common arguments for **ggdist** functions work similarly; e.g. `density`, `align`, `breaks`, `bandwidth`, and `point_interval` arguments.

These function families (except `point_interval()`) also support passing waivers to their optional arguments: if `waiver()` is passed to any of these arguments, their default value (or the most recently-partially-applied non-waiver value) is used instead.

Use the `auto_partial()` function to create new functions that support automatic partial application.

Usage

```
auto_partial(f, name = NULL, waivable = TRUE)
```

Arguments

f	<function> Function to automatically partially-apply.
name	<string> Name of the function, to be used when printing.
waivable	<scalar logical> If TRUE, optional arguments that get passed a waiver() will keep their default value (or whatever non-waiver value has been most recently partially applied for that argument).

Value

A modified version of f that will automatically be partially applied if all of its required arguments are not given.

Examples

```
set.seed(1234)
x = rnorm(100)

# the first required argument, `x`, of the density_ family is the vector
# to calculate a kernel density estimate from. If it is not provided, the
# function is partially applied and returned as-is
density_unbounded()

# we could create a new function that uses half the default bandwidth
density_half_bw = density_unbounded(adjust = 0.5)
density_half_bw

# we can overwrite partially-applied arguments
density_quarter_bw_trimmed = density_half_bw(adjust = 0.25, trim = TRUE)
density_quarter_bw_trimmed

# when we eventually call the function and provide the required argument
# `x`, it is applied using the arguments we have "saved up" so far
density_quarter_bw_trimmed(x)

# create a custom automatically partially applied function
f = auto_partial(function(x, y, z = 3) (x + y) * z)
f()
f(1)
g = f(y = 2)(z = 4)
g
g(1)

# pass waiver() to optional arguments to use existing values
f(z = waiver())(1, 2)  # uses default z = 3
f(z = 4)(z = waiver())(1, 2)  # uses z = 4
```

bandwidth *Bandwidth estimators*

Description

Bandwidth estimators for densities, used in the `bandwidth` argument to density functions (e.g. `density_bounded()`, `density_unbounded()`).

Supports automatic partial function application with waived arguments.

Usage

```
bandwidth_nrd0(x, ...)

bandwidth_nrd(x, ...)

bandwidth_ucv(x, ...)

bandwidth_bcv(x, ...)

bandwidth_SJ(x, ...)

bandwidth_dpi(x, ...)
```

Arguments

`x` <numeric> Vector containing a sample.

`...` Arguments passed on to `stats::bw.SJ`

- `nb` number of bins to use.
- `lower,upper` range over which to minimize. The default is almost always satisfactory. `hmax` is calculated internally from a normal reference bandwidth.
- `method` either `"ste"` ("solve-the-equation") or `"dpi"` ("direct plug-in"). Can be abbreviated.
- `tol` for method `"ste"`, the convergence tolerance for `uniroot`. The default leads to bandwidth estimates with only slightly more than one digit accuracy, which is sufficient for practical density estimation, but possibly not for theoretical simulation studies.

Details

These are loose wrappers around the corresponding `bw.`-prefixed functions in **stats**. See, for example, `bw.SJ()`.

`bandwidth_dpi()`, which is the default bandwidth estimator in **ggdist**, is the Sheather-Jones direct plug-in estimator, i.e. `bw.SJ(..., method = "dpi")`.

With the exception of `bandwidth_nrd0()`, these estimators may fail in some cases, often when a sample contains many duplicates. If they do they will automatically fall back to `bandwidth_nrd0()`

with a warning. However, these failures are typically symptomatic of situations where you should not want to use a kernel density estimator in the first place (e.g. data with duplicates and/or discrete data). In these cases consider using a dotplot (`geom_dots()`) or histogram (`density_histogram()`) instead.

Value

A single number giving the bandwidth

See Also

`density_bounded()`, `density_unbounded()`.

`bin_dots` *Bin data values using a dotplot algorithm*

Description

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

Usage

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

Arguments

`x`	<numeric> *x* values.
`y`	<numeric> *y* values (same length as `x`).
`binwidth`	<scalar numeric> Bin width.
`heightratio`	<scalar numeric> Ratio of bin width to dot height
`stackratio`	<scalar numeric> Ratio of dot height to vertical distance between dot centers
`layout`	<string> The layout method used for the dots. One of:

- "bin" (default): places dots on the off-axis at the midpoint of their bins as in the classic Wilkinson dotplot. This maintains the alignment of rows and columns in the dotplot. This layout is slightly different from the classic Wilkinson algorithm in that: (1) it nudges bins slightly to avoid overlapping bins and (2) if the input data are symmetrical it will return a symmetrical layout.
- "weave": uses the same basic binning approach of "bin", but places dots in the off-axis at their actual positions (unless overlaps = "nudge", in which case overlaps may be nudged out of the way). This maintains the alignment of rows but does not align dots within columns.
- "hex": uses the same basic binning approach of "bin", but alternates placing dots + binwidth/4 or - binwidth/4 in the off-axis from the bin center. This allows hexagonal packing by setting a stackratio less than 1 (something like 0.9 tends to work).
- "swarm": uses the "compactswarm" layout from beeswarm::beeswarm(). Does not maintain alignment of rows or columns, but can be more compact and neat looking, especially for sample data (as opposed to quantile dotplots of theoretical distributions, which may look better with "bin", "weave", or "hex").
- "bar": for discrete distributions, lays out duplicate values in rectangular bars.

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

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

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

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

overlaps
: <string> How to handle overlapping dots or bins in the "bin", "weave", and "hex" layouts (dots never overlap in the "swarm" or "bar" layouts). For the purposes of this argument, dots are only considered to be overlapping if they would be overlapping when dotsize = 1 and stackratio = 1; i.e. if you set those arguments to other values, overlaps may still occur. One of:

- "keep": leave overlapping dots as they are. Dots may overlap (usually only slightly) in the "bin", "weave", and "hex" layouts.
- "nudge": nudge overlapping dots out of the way. Overlaps are avoided using a constrained optimization which minimizes the squared distance of

dots to their desired positions, subject to the constraint that adjacent dots do not overlap.

Value

A data.frame with three columns:

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

See Also

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

Examples

```
library(dplyr)
library(ggplot2)

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

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

---

blur *Blur functions for blurry dot plots*

Description

Methods for constructing blurs, as used in the blur argument to geom_blur_dots() or stat_mcse_dots().

Supports automatic partial function application with waived arguments.

Usage

```
blur_gaussian(x, r, sd)

blur_interval(x, r, sd, .width = 0.95)
```

Arguments

x	<numeric> Vector of positive distances from the center of the dot (assumed to be 0) to evaluate blur function at.
r	<scalar numeric> Radius of the dot that is being blurred.
sd	<scalar numeric> Standard deviation of the dot that is being blurred.
.width	<scalar numeric> For `blur_interval()`, a probability giving the width of the interval.

Details

These functions are passed `x`, `r`, and `sd` when `geom_blur_dots()` draws in order to create a radial gradient representing each dot in the dotplot. They return values between `0` and `1` giving the opacity of the dot at each value of `x`.

`blur_gaussian()` creates a dot with radius `r` that has a Gaussian blur with standard deviation `sd` applied to it. It does this by calculating $\alpha(x; r, \sigma)$, the opacity at distance x from the center of a dot with radius r that has had a Gaussian blur with standard deviation σ = `sd` applied to it:

$$\alpha(x; r, \sigma) = \Phi\left(\frac{x+r}{\sigma}\right) - \Phi\left(\frac{x-r}{\sigma}\right)$$

`blur_interval()` creates an interval-type representation around the dot at 50% opacity, where the interval is a Gaussian quantile interval with mass equal to `.width` and standard deviation `sd`.

Value

A vector with the same length as `x` giving the opacity of the radial gradient representing the dot at each `x` value.

See Also

`geom_blur_dots()` and `stat_mcse_dots()` for geometries making use of `blur` functions.

Examples

```
# see examples in geom_blur_dots()
```

---

bounder_cdf — *Estimate bounds of a distribution using the CDF of its order statistics*

Description

Estimate the bounds of the distribution a sample came from using the CDF of the order statistics of the sample. Use with the `bounder` argument to `density_bounded()`.

Supports automatic partial function application with waived arguments.

Usage

```
bounder_cdf(x, p = 0.01)
```

Arguments

x	<numeric> Sample to estimate the bounds of.
p	<scalar numeric> in $[0, 1]$: Percentile of the order statistic distribution to use as the estimate. `p = 1` will return `range(x)`; `p = 0.5` will give the median estimate, `p = 0` will give a very wide estimate (effectively treating the distribution as unbounded when used with `density_bounded()`).

Details

`bounder_cdf()` uses the distribution of the order statistics of X to estimate where the first and last order statistics (i.e. the min and max) of this distribution would be, assuming the sample `x` is the distribution. Then, it adjusts the boundary outwards from `min(x)` (or `max(x)`) by the distance between `min(x)` (or `max(x)`) and the nearest estimated order statistic.

Taking X = `x`, the distributions of the first and last order statistics are:

$$\begin{array}{rcl} F_{X_{(1)}}(x) & = & 1 - [1 - F_X(x)]^n \\ F_{X_{(n)}}(x) & = & F_X(x)^n \end{array}$$

Re-arranging, we can get the inverse CDFs (quantile functions) of each order statistic in terms of the quantile function of X (which we can estimate from the data), giving us an estimate for the minimum and maximum order statistic:

$$\begin{array}{rcccl} \hat{x_1} & = & F^{-1}_{X_{(1)}}(p) & = & F^{-1}_X\left[1 - (1 - p)^{1/n}\right] \\ \hat{x_n} & = & F^{-1}_{X_{(n)}}(p) & = & F^{-1}_X\left[p^{1/n}\right] \end{array}$$

Then the estimated bounds are:

$$[2\min(x) - \hat{x_1}, 2\max(x) - \hat{x_n}]$$

These bounds depend on p, the percentile of the distribution of the order statistic used to form the estimate. While $p = 0.5$ (the median) might be a reasonable choice (and gives results similar to `bounder_cooke()`), this tends to be a bit too aggressive in "detecting" bounded distributions, especially in small sample sizes. Thus, we use a default of $p = 0.01$, which tends to be very conservative in small samples (in that it usually gives results roughly equivalent to an unbounded distribution), but which still performs well on bounded distributions when sample sizes are larger (in the thousands).

Value

A length-2 numeric vector giving an estimate of the minimum and maximum bounds of the distribution that `x` came from.

See Also

The bounder argument to density_bounded().

Other bounds estimators: bounder_cooke(), bounder_range()

---

bounder_cooke *Estimate bounds of a distribution using Cooke's method*

Description

Estimate the bounds of the distribution a sample came from using Cooke's method. Use with the bounder argument to density_bounded().

Supports automatic partial function application with waived arguments.

Usage

```
bounder_cooke(x)
```

Arguments

x	<numeric> Sample to estimate the bounds of.

Details

Estimate the bounds of a distribution using the method from Cooke (1979); i.e. method 2.3 from Loh (1984). These bounds are:

$$\begin{bmatrix} 2X_{(1)} - \sum_{i=1}^{n} \left[\left(1 - \frac{i-1}{n}\right)^n - \left(1 - \frac{i}{n}\right)^n \right] X_{(i)} \\ 2X_{(n)} - \sum_{i=1}^{n} \left[\left(1 - \frac{n-i}{n}\right)^n - \left(1 - \frac{n+1-i}{n}\right)^n \right] X_{(i)} \end{bmatrix}$$

Where $X_{(i)}$ is the ith order statistic of x (i.e. its ith-smallest value).

Value

A length-2 numeric vector giving an estimate of the minimum and maximum bounds of the distribution that x came from.

References

Cooke, P. (1979). Statistical inference for bounds of random variables. *Biometrika* 66(2), 367–374. doi:10.1093/biomet/66.2.367.

Loh, W. Y. (1984). Estimating an endpoint of a distribution with resampling methods. *The Annals of Statistics* 12(4), 1543–1550. doi:10.1214/aos/1176346811

See Also

The bounder argument to density_bounded().

Other bounds estimators: bounder_cdf(), bounder_range()

bounder_range *Estimate bounds of a distribution using the range of the sample*

Description

Estimate the bounds of the distribution a sample came from using the range of the sample. Use with the `bounder` argument to `density_bounded()`.

Supports automatic partial function application with waived arguments.

Usage

```
bounder_range(x)
```

Arguments

`x`	<numeric> Sample to estimate the bounds of.

Details

Estimate the bounds of a distribution using `range(x)`.

Value

A length-2 numeric vector giving an estimate of the minimum and maximum bounds of the distribution that `x` came from.

See Also

The `bounder` argument to `density_bounded()`.

Other bounds estimators: `bounder_cdf()`, `bounder_cooke()`

breaks *Break (bin) selection algorithms for histograms*

Description

Methods for determining breaks (bins) in histograms, as used in the `breaks` argument to `density_histogram()`.

Supports automatic partial function application with waived arguments.

Usage

```
breaks_fixed(x, weights = NULL, width = 1)

breaks_Sturges(x, weights = NULL)

breaks_Scott(x, weights = NULL)

breaks_FD(x, weights = NULL, digits = 5)

breaks_quantiles(x, weights = NULL, max_n = "Scott", min_width = 0.5)
```

Arguments

`x`	<numeric> Sample values.
`weights`	<numeric | NULL> Optional weights to apply to `x`, which will be normalized to sum to 1.
`width`	<scalar numeric> For `breaks_fixed()`, the desired bin width.
`digits`	<scalar numeric> For `breaks_FD()`, the number of significant digits to keep when rounding in the Freedman-Diaconis algorithm. For an explanation of this parameter, see the documentation of the corresponding parameter in `grDevices::nclass.FD()`.
`max_n`	<scalar numeric | function | string> For `breaks_quantiles()`, either a scalar numeric giving the maximum number of bins, or another breaks function (or string giving the suffix of the name of a function prefixed with `"breaks_"`) that will return the maximum number of bins. `breaks_quantiles()` will construct *at most* `max_n` bins.
`min_width`	<scalar numeric> For `breaks_quantiles()`, a numeric between `0` and `1` giving the minimum bin width as a proportion of `diff(range(x)) / max_n`.

Details

These functions take a sample and its weights and return a value suitable for the `breaks` argument to `density_histogram()` that will determine the histogram breaks.

- `breaks_fixed()` allows you to manually specify a fixed bin width.
- `breaks_Sturges()`, `breaks_Scott()`, and `breaks_FD()` implement weighted versions of their corresponding base functions. They return a scalar numeric giving the number of bins. See `nclass.Sturges()`, `nclass.scott()`, and `nclass.FD()`.
- `breaks_quantiles()` constructs irregularly-sized bins using `max_n + 1` (possibly weighted) quantiles of `x`. The final number of bins is *at most* `max_n`, as small bins (ones whose bin width is less than half the range of the data divided by `max_n` times `min_width`) will be merged into adjacent bins.

Value

Either a single number (giving the number of bins) or a vector giving the edges between bins.

See Also

density_histogram(), align

Examples

```
library(ggplot2)

set.seed(1234)
x = rnorm(2000, 1, 2)

# Let's compare the different break-selection algorithms on this data:
ggplot(data.frame(x), aes(x)) +
  stat_slab(
    aes(y = "breaks_fixed(width = 0.5)"),
    density = "histogram",
    breaks = breaks_fixed(width = 0.5),
    outline_bars = TRUE,
    color = "black",
  ) +
  stat_slab(
    aes(y = "breaks_Sturges()\nor 'Sturges'"),
    density = "histogram",
    breaks = "Sturges",
    outline_bars = TRUE,
    color = "black",
  ) +
  stat_slab(
    aes(y = "breaks_Scott()\nor 'Scott'"),
    density = "histogram",
    breaks = "Scott",
    outline_bars = TRUE,
    color = "black",
  ) +
  stat_slab(
    aes(y = "breaks_FD()\nor 'FD'"),
    density = "histogram",
    breaks = "FD",
    outline_bars = TRUE,
    color = "black",
  ) +
  stat_slab(
    aes(y = "breaks_quantiles()\nor 'quantiles'"),
    density = "histogram",
    breaks = "quantiles",
    outline_bars = TRUE,
    color = "black",
  ) +
  geom_point(aes(y = 0.7), alpha = 0.5) +
  labs(
    subtitle = "ggdist::stat_slab(density = 'histogram', ...)",
    y = "breaks =",
    x = NULL
```

```
)
```

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

Description

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

Usage

```
curve_interval(
  .data,
  ...,
  .along = NULL,
  .width = 0.5,
  na.rm = FALSE,
  .interval = c("mhd", "mbd", "bd", "bd-mbd")
)

## S3 method for class 'matrix'
curve_interval(
  .data,
  ...,
  .along = NULL,
  .width = 0.5,
  na.rm = FALSE,
  .interval = c("mhd", "mbd", "bd", "bd-mbd")
)

## S3 method for class 'rvar'
curve_interval(
  .data,
  ...,
  .along = NULL,
  .width = 0.5,
  na.rm = FALSE,
  .interval = c("mhd", "mbd", "bd", "bd-mbd")
)

## S3 method for class 'data.frame'
curve_interval(
  .data,
  ...,
  .along = NULL,
```

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

Arguments

`.data` <data.frame | rvar | matrix> One of:

- A data frame (or grouped data frame as returned by `dplyr::group_by()`) that contains draws to summarize.
- A posterior::rvar vector.
- A matrix; in which case the first dimension should be draws and the second dimension values of the curve.

`...` <bare language> 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 numeric columns, list columns containing numeric vectors, or `posterior::rvar()`s.

`.along` <tidyselect> Which columns are the input values to the function describing the curve (e.g., the "x" values). Intervals are calculated jointly with respect to these variables, conditional on all other grouping variables in the data frame. The default (`NULL`) causes `curve_interval()` to use all grouping variables in the input data frame as the value for `.along`, which will generate the most conservative intervals. However, if you want to calculate intervals for some function `y = f(x)` conditional on some other variable(s) (say, conditional on a factor `g`), you would group by `g`, then use `.along = x` to calculate intervals jointly over `x` conditional on `g`. To avoid selecting any variables as input values to the function describing the curve, use `character()`; this will produce conditional intervals only (the result in this case should be very similar to `median_qi()`). Currently only supported when `.data` is a data frame.

`.width` <numeric> 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).

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

`.interval` <string> The method used to calculate the intervals. Currently, all methods rank the curves using some measure of *data depth*, then create envelopes containing the `.width%` "deepest" curves. Available methods are:

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

.simple_names <scalar logical> 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. When FALSE and .data is a data frame, names the lower and upper intervals for each column x x.lower and x.upper.

.exclude <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 **ggdist** and **tidybayes**.

Details

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

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

Value

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

Author(s)

Matthew Kay

References

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

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

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

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

See Also

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

Examples

```
library(dplyr)
library(ggplot2)

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

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


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

cut_cdf_qi *Categorize values from a CDF into quantile intervals*

Description

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

Usage

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

Arguments

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

`.width` <numeric> Vector of probabilities to use that determine the widths of the resulting intervals.

`labels` <character | function | NULL> One of:

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

Value

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

See Also

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

Examples

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

theme_set(theme_ggdist())
```

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

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

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

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

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

density_bounded *Bounded density estimator using the reflection method*

Description

Bounded density estimator using the reflection method.

Supports automatic partial function application with waived arguments.

Usage

```
density_bounded(
  x,
  weights = NULL,
  n = 501,
  bandwidth = "dpi",
  adjust = 1,
  kernel = "gaussian",
  trim = TRUE,
  bounds = c(NA, NA),
  bounder = "cdf",
  adapt = 1,
  na.rm = FALSE,
  ...,
  range_only = FALSE
)
```

Arguments

`x` <numeric> Sample to compute a density estimate for.

`weights` <numeric | NULL> Optional weights to apply to `x`.

`n` <scalar numeric> The number of grid points to evaluate the density estimator at.

`bandwidth` <scalar numeric | function | string> Bandwidth of the density estimator. One of:

- a numeric: the bandwidth, as the standard deviation of the kernel
- a function: a function taking `x` (the sample) and returning the bandwidth
- a string: the suffix of the name of a function starting with `"bandwidth_"` that will be used to determine the bandwidth. See bandwidth for a list.

`adjust` <scalar numeric> Value to multiply the bandwidth of the density estimator by. Default `1`.

`kernel` <string> The smoothing kernel to be used. This must partially match one of `"gaussian"`, `"rectangular"`, `"triangular"`, `"epanechnikov"`, `"biweight"`, `"cosine"`, or `"optcosine"`. See `stats::density()`.

`trim` <scalar logical> Should the density estimate be trimmed to the range of the data? Default `TRUE`.

`bounds` <length-2 numeric> Min and max bounds. If a bound is `NA`, then that bound is estimated from the data using the method specified by `bounder`.

`bounder` <function | string> Method to use to find missing (`NA`) bounds. A function that takes a numeric vector of values and returns a length-2 vector of the estimated lower and upper bound of the distribution. Can also be a string giving the suffix of the name of such a function that starts with `"bounder_"`. Useful values include:

- "cdf": Use the CDF of the the minimum and maximum order statistics of the sample to estimate the bounds. See bounder_cdf().
- "cooke": Use the method from Cooke (1979); i.e. method 2.3 from Loh (1984). See bounder_cooke().
- "range": Use the range of x (i.e the min or max). See bounder_range().

adapt
: <positive integer> (**very experimental**) The name and interpretation of this argument are subject to change without notice. If adapt > 1, uses an adaptive approach to calculate the density. First, uses the adaptive bandwidth algorithm of Abramson (1982) to determine local (pointwise) bandwidths, then groups these bandwidths into adapt groups, then calculates and sums the densities from each group. You can set this to a very large number (e.g. Inf) for a fully adaptive approach, but this will be very slow; typically something around 100 yields nearly identical results.

na.rm
: <scalar logical> Should missing (NA) values in x be removed?

...
: Additional arguments (ignored).

range_only
: <scalar logical> If TRUE, the range of the output of this density estimator is computed and is returned in the $x element of the result, and c(NA, NA) is returned in $y. This gives a faster way to determine the range of the output than density_XXX(n = 2).

Value

An object of class "density", mimicking the output format of stats::density(), with the following components:

- x: The grid of points at which the density was estimated.
- y: The estimated density values.
- bw: The bandwidth.
- n: The sample size of the x input argument.
- call: The call used to produce the result, as a quoted expression.
- data.name: The deparsed name of the x input argument.
- has.na: Always FALSE (for compatibility).
- cdf: Values of the (possibly weighted) empirical cumulative distribution function at x. See weighted_ecdf().

This allows existing methods for density objects, like print() and plot(), to work if desired. This output format (and in particular, the x and y components) is also the format expected by the density argument of the stat_slabinterval() and the smooth_ family of functions.

References

Cooke, P. (1979). Statistical inference for bounds of random variables. *Biometrika* 66(2), 367–374. doi:10.1093/biomet/66.2.367.

Loh, W. Y. (1984). Estimating an endpoint of a distribution with resampling methods. *The Annals of Statistics* 12(4), 1543–1550. doi:10.1214/aos/1176346811

See Also

Other density estimators: density_histogram(), density_unbounded()

Examples

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

# For compatibility with existing code, the return type of density_bounded()
# is the same as stats::density(), ...
set.seed(123)
x = rbeta(5000, 1, 3)
d = density_bounded(x)
d

# ... thus, while designed for use with the `density` argument of
# stat_slabinterval(), output from density_bounded() can also be used with
# base::plot():
plot(d)

# here we'll use the same data as above, but pick either density_bounded()
# or density_unbounded() (which is equivalent to stats::density()). Notice
# how the bounded density (green) is biased near the boundary of the support,
# while the unbounded density is not.
data.frame(x) %>%
  ggplot() +
  stat_slab(
    aes(xdist = dist), data = data.frame(dist = dist_beta(1, 3)),
    alpha = 0.25
  ) +
  stat_slab(aes(x), density = "bounded", fill = NA, color = "#d95f02", alpha = 0.5) +
  stat_slab(aes(x), density = "unbounded", fill = NA, color = "#1b9e77", alpha = 0.5) +
  scale_thickness_shared() +
  theme_ggdist()

# We can also supply arguments to the density estimators by using their
# full function names instead of the string suffix; e.g. we can supply
# the exact bounds of c(0,1) rather than using the bounds of the data.
data.frame(x) %>%
  ggplot() +
  stat_slab(
    aes(xdist = dist), data = data.frame(dist = dist_beta(1, 3)),
    alpha = 0.25
  ) +
  stat_slab(
    aes(x), fill = NA, color = "#d95f02", alpha = 0.5,
    density = density_bounded(bounds = c(0,1))
  ) +
  scale_thickness_shared() +
  theme_ggdist()
```

density_histogram *Histogram density estimator*

Description

Histogram density estimator.

Supports automatic partial function application with waived arguments.

Usage

```
density_histogram(
  x,
  weights = NULL,
  breaks = "Scott",
  align = "none",
  outline_bars = FALSE,
  right_closed = TRUE,
  outermost_closed = TRUE,
  na.rm = FALSE,
  ...,
  range_only = FALSE
)
```

Arguments

x <numeric> Sample to compute a density estimate for.

weights <numeric | NULL> Optional weights to apply to x.

breaks <numeric | function | string> Determines the breakpoints defining bins. Default "Scott". Similar to (but not exactly the same as) the breaks argument to graphics::hist(). One of:

- A scalar (length-1) numeric giving the number of bins
- A vector numeric giving the breakpoints between histogram bins
- A function taking x and weights and returning either the number of bins or a vector of breakpoints
- A string giving the suffix of a function that starts with "breaks_". **ggdist** provides weighted implementations of the "Sturges", "Scott", and "FD" break-finding algorithms from graphics::hist(), as well as breaks_fixed() for manually setting the bin width. See breaks.

For example, breaks = "Sturges" will use the breaks_Sturges() algorithm, breaks = 9 will create 9 bins, and breaks = breaks_fixed(width = 1) will set the bin width to 1.

align <scalar numeric | function | string> Determines how to align the breakpoints defining bins. Default "none" (performs no alignment). One of:

- A scalar (length-1) numeric giving an offset that is subtracted from the breaks. The offset must be between 0 and the bin width.

- A function taking a sorted vector of `breaks` (bin edges) and returning an offset to subtract from the breaks.
- A string giving the suffix of a function that starts with `"align_"` used to determine the alignment, such as `align_none()`, `align_boundary()`, or `align_center()`.

For example, `align = "none"` will provide no alignment, `align = align_center(at = 0)` will center a bin on `0`, and `align = align_boundary(at = 0)` will align a bin edge on `0`.

`outline_bars` <scalar logical> Should outlines in between the bars (i.e. density values of 0) be included?

`right_closed` <scalar logical> Should the right edge of each bin be closed? For a bin with endpoints L and U:

- if `TRUE`, use $(L, U]$: the interval containing all x such that $L < x \leq U$.
- if `FALSE`, use $[L, U)$: the interval containing all x such that $L \leq x < U$.

Equivalent to the `right` argument of `hist()` or the `left.open` argument of `findInterval()`.

`outermost_closed`

<scalar logical> Should values on the edges of the outermost (first or last) bins always be included in those bins? If `TRUE`, the first edge (when `right_closed = TRUE`) or the last edge (when `right_closed = FALSE`) is treated as closed.

Equivalent to the `include.lowest` argument of `hist()` or the `rightmost.closed` argument of `findInterval()`.

`na.rm` <scalar logical> Should missing (`NA`) values in `x` be removed?

`...` Additional arguments (ignored).

`range_only` <scalar logical> If `TRUE`, the range of the output of this density estimator is computed and is returned in the `$x` element of the result, and `c(NA, NA)` is returned in `$y`. This gives a faster way to determine the range of the output than `density_XXX(n = 2)`.

Value

An object of class `"density"`, mimicking the output format of `stats::density()`, with the following components:

- `x`: The grid of points at which the density was estimated.
- `y`: The estimated density values.
- `bw`: The bandwidth.
- `n`: The sample size of the `x` input argument.
- `call`: The call used to produce the result, as a quoted expression.
- `data.name`: The deparsed name of the `x` input argument.
- `has.na`: Always `FALSE` (for compatibility).
- `cdf`: Values of the (possibly weighted) empirical cumulative distribution function at `x`. See `weighted_ecdf()`.

This allows existing methods for density objects, like `print()` and `plot()`, to work if desired. This output format (and in particular, the `x` and `y` components) is also the format expected by the `density` argument of the `stat_slabinterval()` and the `smooth_` family of functions.

See Also

Other density estimators: density_bounded(), density_unbounded()

Examples

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

# For compatibility with existing code, the return type of density_unbounded()
# is the same as stats::density(), ...
set.seed(123)
x = rbeta(5000, 1, 3)
d = density_histogram(x)
d

# ... thus, while designed for use with the `density` argument of
# stat_slabinterval(), output from density_histogram() can also be used with
# base::plot():
plot(d)

# here we'll use the same data as above with stat_slab():
data.frame(x) %>%
  ggplot() +
  stat_slab(
    aes(xdist = dist), data = data.frame(dist = dist_beta(1, 3)),
    alpha = 0.25
  ) +
  stat_slab(aes(x), density = "histogram", fill = NA, color = "#d95f02", alpha = 0.5) +
  scale_thickness_shared() +
  theme_ggdist()
```

---

density_unbounded   *Unbounded density estimator*

Description

Unbounded density estimator using stats::density().

Supports automatic partial function application with waived arguments.

Usage

```
density_unbounded(
  x,
  weights = NULL,
  n = 501,
  bandwidth = "dpi",
  adjust = 1,
```

```
    kernel = "gaussian",
    trim = TRUE,
    adapt = 1,
    na.rm = FALSE,
    ...,
    range_only = FALSE
)
```

Arguments

`x`	<numeric> Sample to compute a density estimate for.
`weights`	<numeric | NULL> Optional weights to apply to `x`.
`n`	<scalar numeric> The number of grid points to evaluate the density estimator at.
`bandwidth`	<scalar numeric | function | string> Bandwidth of the density estimator. One of: • a numeric: the bandwidth, as the standard deviation of the kernel • a function: a function taking `x` (the sample) and returning the bandwidth • a string: the suffix of the name of a function starting with `"bandwidth_"` that will be used to determine the bandwidth. See bandwidth for a list.
`adjust`	<scalar numeric> Value to multiply the bandwidth of the density estimator by. Default `1`.
`kernel`	<string> The smoothing kernel to be used. This must partially match one of `"gaussian"`, `"rectangular"`, `"triangular"`, `"epanechnikov"`, `"biweight"`, `"cosine"`, or `"optcosine"`. See `stats::density()`.
`trim`	<scalar logical> Should the density estimate be trimmed to the range of the data? Default `TRUE`.
`adapt`	<positive integer> (**very experimental**) The name and interpretation of this argument are subject to change without notice. If `adapt > 1`, uses an adaptive approach to calculate the density. First, uses the adaptive bandwidth algorithm of Abramson (1982) to determine local (pointwise) bandwidths, then groups these bandwidths into `adapt` groups, then calculates and sums the densities from each group. You can set this to a very large number (e.g. `Inf`) for a fully adaptive approach, but this will be very slow; typically something around 100 yields nearly identical results.
`na.rm`	<scalar logical> Should missing (`NA`) values in `x` be removed?
`...`	Additional arguments (ignored).
`range_only`	<scalar logical> If `TRUE`, the range of the output of this density estimator is computed and is returned in the `$x` element of the result, and `c(NA, NA)` is returned in `$y`. This gives a faster way to determine the range of the output than `density_XXX(n = 2)`.

Value

An object of class `"density"`, mimicking the output format of `stats::density()`, with the following components:

- `x`: The grid of points at which the density was estimated.

- y: The estimated density values.
- bw: The bandwidth.
- n: The sample size of the x input argument.
- call: The call used to produce the result, as a quoted expression.
- data.name: The deparsed name of the x input argument.
- has.na: Always FALSE (for compatibility).
- cdf: Values of the (possibly weighted) empirical cumulative distribution function at x. See weighted_ecdf().

This allows existing methods for density objects, like print() and plot(), to work if desired. This output format (and in particular, the x and y components) is also the format expected by the density argument of the stat_slabinterval() and the smooth_ family of functions.

See Also

Other density estimators: density_bounded(), density_histogram()

Examples

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

# For compatibility with existing code, the return type of density_unbounded()
# is the same as stats::density(), ...
set.seed(123)
x = rbeta(5000, 1, 3)
d = density_unbounded(x)
d

# ... thus, while designed for use with the `density` argument of
# stat_slabinterval(), output from density_unbounded() can also be used with
# base::plot():
plot(d)

# here we'll use the same data as above, but pick either density_bounded()
# or density_unbounded() (which is equivalent to stats::density()). Notice
# how the bounded density (green) is biased near the boundary of the support,
# while the unbounded density is not.
data.frame(x) %>%
  ggplot() +
  stat_slab(
    aes(xdist = dist), data = data.frame(dist = dist_beta(1, 3)),
    alpha = 0.25
  ) +
  stat_slab(aes(x), density = "bounded", fill = NA, color = "#d95f02", alpha = 0.5) +
  stat_slab(aes(x), density = "unbounded", fill = NA, color = "#1b9e77", alpha = 0.5) +
  scale_thickness_shared() +
  theme_ggdist()
```

`find_dotplot_binwidth` *Dynamically select a good bin width for a dotplot*

Description

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

Usage

```
find_dotplot_binwidth(
  x,
  maxheight,
  heightratio = 1,
  stackratio = 1,
  layout = c("bin", "weave", "hex", "swarm", "bar")
)
```

Arguments

`x`	<numeric> Data values.
`maxheight`	<scalar numeric> Maximum height of the dotplot.
`heightratio`	<scalar numeric> Ratio of bin width to dot height.
`stackratio`	<scalar numeric> Ratio of dot height to vertical distance between dot centers
`layout`	<string> The layout method used for the dots. One of: • `"bin"` (default): places dots on the off-axis at the midpoint of their bins as in the classic Wilkinson dotplot. This maintains the alignment of rows and columns in the dotplot. This layout is slightly different from the classic Wilkinson algorithm in that: (1) it nudges bins slightly to avoid overlapping bins and (2) if the input data are symmetrical it will return a symmetrical layout. • `"weave"`: uses the same basic binning approach of `"bin"`, but places dots in the off-axis at their actual positions (unless `overlaps = "nudge"`, in which case overlaps may be nudged out of the way). This maintains the alignment of rows but does not align dots within columns. • `"hex"`: uses the same basic binning approach of `"bin"`, but alternates placing dots `+ binwidth/4` or `- binwidth/4` in the off-axis from the bin center. This allows hexagonal packing by setting a `stackratio` less than 1 (something like `0.9` tends to work). • `"swarm"`: uses the `"compactswarm"` layout from `beeswarm::beeswarm()`. Does not maintain alignment of rows or columns, but can be more compact and neat looking, especially for sample data (as opposed to quantile dotplots of theoretical distributions, which may look better with `"bin"`, `"weave"`, or `"hex"`). • `"bar"`: for discrete distributions, lays out duplicate values in rectangular bars.

Details

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

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

Value

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

See Also

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

Examples

```
library(dplyr)
library(ggplot2)

x = qnorm(ppoints(20))
binwidth = find_dotplot_binwidth(x, maxheight = 4, heightratio = 1)
binwidth

bin_df = bin_dots(x = x, y = 0, binwidth = binwidth, heightratio = 1)
bin_df

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

---

geom_blur_dots *Blurry dot plot (geom)*

Description

Variant of geom_dots() for creating blurry dotplots. Accepts an sd aesthetic that gives the standard deviation of the blur applied to the dots. Requires a graphics engine supporting radial gradients. Unlike geom_dots(), this geom only supports circular and square shapes.

Usage

```
geom_blur_dots(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  blur = "gaussian",
  binwidth = NA,
  dotsize = 1.07,
  stackratio = 1,
  layout = "bin",
  overlaps = "nudge",
  smooth = "none",
  overflow = "warn",
  verbose = FALSE,
  orientation = NA,
  subguide = "slab",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

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

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

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

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

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

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

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

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

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

blur
: <function | string> Blur function to apply to dots. One of:
  - A function that takes a numeric vector of distances from the dot center, the dot radius, and the standard deviation of the blur and returns a vector of opacities in $[0, 1]$, such as `blur_gaussian()` or `blur_interval()`.
  - A string indicating what blur function to use, as the suffix to a function name starting with `blur_`; e.g. `"gaussian"` (the default) applies `blur_gaussian()`.

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

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

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

stackratio
: <scalar numeric> The distance between the center of the dots in the same stack relative to the dot height. The default, `1`, makes dots in the same stack just touch each other.

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

- "weave": uses the same basic binning approach of "bin", but places dots in the off-axis at their actual positions (unless overlaps = "nudge", in which case overlaps may be nudged out of the way). This maintains the alignment of rows but does not align dots within columns.
- "hex": uses the same basic binning approach of "bin", but alternates placing dots + binwidth/4 or - binwidth/4 in the off-axis from the bin center. This allows hexagonal packing by setting a stackratio less than 1 (something like 0.9 tends to work).
- "swarm": uses the "compactswarm" layout from beeswarm::beeswarm(). Does not maintain alignment of rows or columns, but can be more compact and neat looking, especially for sample data (as opposed to quantile dotplots of theoretical distributions, which may look better with "bin", "weave", or "hex").
- "bar": for discrete distributions, lays out duplicate values in rectangular bars.

overlaps
: <string> How to handle overlapping dots or bins in the "bin", "weave", and "hex" layouts (dots never overlap in the "swarm" or "bar" layouts). For the purposes of this argument, dots are only considered to be overlapping if they would be overlapping when dotsize = 1 and stackratio = 1; i.e. if you set those arguments to other values, overlaps may still occur. One of:

- "keep": leave overlapping dots as they are. Dots may overlap (usually only slightly) in the "bin", "weave", and "hex" layouts.
- "nudge": nudge overlapping dots out of the way. Overlaps are avoided using a constrained optimization which minimizes the squared distance of dots to their desired positions, subject to the constraint that adjacent dots do not overlap.

smooth
: <function | string> Smoother to apply to dot positions. One of:

- A function that takes a numeric vector of dot positions and returns a smoothed version of that vector, such as smooth_bounded(), smooth_unbounded(), smooth_discrete(), or smooth_bar()‘.
- A string indicating what smoother to use, as the suffix to a function name starting with smooth_; e.g. "none" (the default) applies smooth_none(), which simply returns the given vector without applying smoothing.

Smoothing is most effective when the smoother is matched to the support of the distribution; e.g. using smooth_bounded(bounds = ...).

overflow
: <string> How to handle overflow of dots beyond the extent of the geom when a minimum binwidth (or an exact binwidth) is supplied. One of:

- "keep": Keep the overflow, drawing dots outside the geom bounds.
- "warn": Keep the overflow, but produce a warning suggesting solutions, such as setting binwidth = NA or overflow = "compress".
- "compress": Compress the layout. Reduces the binwidth to the size necessary to keep the dots within bounds, then adjusts stackratio and dotsize so that the apparent dot size is the user-specified minimum binwidth times the user-specified dotsize.

If you find the default layout has dots that are too small, and you are okay with dots overlapping, consider setting `overflow = "compress"` and supplying an exact or minimum dot size using `binwidth`.

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

`orientation` <string> Whether this geom is drawn horizontally or vertically. One of:

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

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

`subguide` <function | string> Sub-guide used to annotate the `thickness` scale. One of:

- A function that takes a `scale` argument giving a ggplot2::Scale object and an `orientation` argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as `subguide_axis()` (to draw a traditional axis) or `subguide_none()` (to draw no annotation). See `subguide_axis()` for a list of possibilities and examples.
- A string giving the name of such a function when prefixed with `"subguide_"`; e.g. `"axis"` or `"none"`. The values `"slab"`, `"dots"`, and `"spike"` use the default subguide for their geom families (no subguide), which can be modified by setting `subguide_slab`, `subguide_dots`, or `subguide_spike`; see the documentation for those functions.

`na.rm` <scalar logical> 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

The *dots* family of stats and geoms are similar to `ggplot2::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 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)

Stats and geoms in this family include:

- `geom_dots()`: dotplots on raw data. Ensures the dotplot fits within available space by reducing the size of the dots automatically (may result in very small dots).
- `geom_swarm()` and `geom_weave()`: dotplots on raw data with defaults intended to create "beeswarm" plots. Used `side = "both"` by default, and sets the default dot size to the same size as `geom_point()` (`binwidth = unit(1.5, "mm")`), allowing dots to overlap instead of getting very small.
- `stat_dots()`: dotplots on raw data, **distributional** objects, and `posterior::rvar()`s
- `geom_dotsinterval()`: dotplot + interval plots on raw data with already-calculated intervals (rarely useful directly).
- `stat_dotsinterval()`: dotplot + interval plots on raw data, **distributional** objects, and `posterior::rvar()`s (will calculate intervals for you).
- `geom_blur_dots()`: blurry dotplots that allow the standard deviation of a blur applied to each dot to be specified using the `sd` aesthetic.
- `stat_mcse_dots()`: blurry dotplots of quantiles using the Monte Carlo Standard Error of each quantile.

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

Value

A ggplot2::Geom representing a blurry dot geometry which can be added to a `ggplot()` object.

Aesthetics

The dots+interval `stat`s and `geom`s have a wide variety of aesthetics that control the appearance of their three sub-geometries: the **dots** (aka the **slab**), the **point**, and the **interval**.

Positional aesthetics

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

Dots-specific (aka Slab-specific) aesthetics

- `sd`: The standard deviation (in data units) of the blur associated with each dot.
- `order`: The order in which data points are stacked within bins. Can be used to create the effect of "stacked" dots by ordering dots according to a discrete variable. If omitted (`NULL`), the value of the data points themselves are used to determine stacking order. Only applies when `layout` is `"bin"` or `"hex"`, as the other layout methods fully determine both *x* and *y* positions.
- `side`: Which side to place the slab on. `"topright"`, `"top"`, and `"right"` are synonyms which cause the slab to be drawn on the top or the right depending on if `orientation` is `"horizontal"` or `"vertical"`. `"bottomleft"`, `"bottom"`, and `"left"` are synonyms which cause the slab to be drawn on the bottom or the left depending on if `orientation` is `"horizontal"` or `"vertical"`. `"topleft"` causes the slab to be drawn on the top or the left, and `"bottomright"` causes the slab to be drawn on the bottom or the right. `"both"` draws the slab mirrored on both sides (as in a violin plot).
- `scale`: What proportion of the region allocated to this geom to use to draw the slab. If `scale = 1`, slabs that use the maximum range will just touch each other. Default is `0.9` to leave some space between adjacent slabs. For a comprehensive discussion and examples of slab scaling and normalization, see the `thickness` scale article.
- `justification`: Justification of the interval relative to the slab, where `0` indicates bottom/left justification and `1` indicates top/right justification (depending on `orientation`). If `justification` is `NULL` (the default), then it is set automatically based on the value of `side`: when `side` is `"top"`/`"right"` `justification` is set to `0`, when `side` is `"bottom"`/`"left"` `justification` is set to `1`, and when `side` is `"both"` `justification` is set to 0.5.
- `datatype`: When using composite geoms directly without a `stat` (e.g. `geom_slabinterval()`), `datatype` is used to indicate which part of the geom a row in the data targets: rows with `datatype = "slab"` target the slab portion of the geometry and rows with `datatype = "interval"` target the interval portion of the geometry. This is set automatically when using ggdist `stat`s.

Interval-specific aesthetics

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

Point-specific aesthetics

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

Color aesthetics

- `colour`: (or `color`) The color of the **interval** and **point** sub-geometries. Use the `slab_color`, `interval_color`, or `point_color` aesthetics (below) to set sub-geometry colors separately.
- `fill`: The fill color of the **slab** and **point** sub-geometries. Use the `slab_fill` or `point_fill` aesthetics (below) to set sub-geometry colors separately.
- `alpha`: The opacity of the **slab**, **interval**, and **point** sub-geometries. Use the `slab_alpha`, `interval_alpha`, or `point_alpha` aesthetics (below) to set sub-geometry colors separately.

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

Line aesthetics

- linewidth: Width of the line used to draw the **interval** (except with geom_slab(): then it is the width of the **slab**). With composite geometries including an interval and slab, use slab_linewidth to set the line width of the **slab** (see below). For **interval**, raw linewidth values are transformed according to the interval_size_domain and interval_size_range parameters of the geom (see above).
- size: Determines the size of the **point**. If linewidth is not provided, size will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only size and not linewidth). Raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the point_size aesthetic (below) to set sub-geometry size directly without applying the effects of interval_size_domain, interval_size_range, and fatten_point.
- stroke: Width of the outline around the **point** sub-geometry.
- linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the slab_linetype or interval_linetype aesthetics (below) to set sub-geometry line types separately.

Slab-specific color and line override aesthetics

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

Interval-specific color and line override aesthetics

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

Point-specific color and line override aesthetics

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

Deprecated aesthetics

- `slab_size`: Use `slab_linewidth`.
- `interval_size`: Use `interval_linewidth`.

Other aesthetics (these work as in standard geoms)

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

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

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 `geom_dotsinterval()` for the geometry this shortcut is based on.

See `vignette("dotsinterval")` for a variety of examples of use.

Other dotsinterval geoms: `geom_dots()`, `geom_dotsinterval()`, `geom_swarm()`, `geom_weave()`

Examples

```
library(dplyr)
library(ggplot2)

theme_set(theme_ggdist())

set.seed(1234)
x = rnorm(1000)

# manually calculate quantiles and their MCSE
# this could also be done more succinctly with stat_mcse_dots()
p = ppoints(100)
df = data.frame(
  q = quantile(x, p),
  se = posterior::mcse_quantile(x, p)
)

df %>%
  ggplot(aes(x = q, sd = se)) +
```

```
    geom_blur_dots()

df %>%
  ggplot(aes(x = q, sd = se)) +
  # or blur = blur_interval(.width = .95) to set the interval width
  geom_blur_dots(blur = "interval")
```

`geom_dots` *Dot plot (shortcut geom)*

Description

Shortcut version of `geom_dotsinterval()` for creating dot plots. Geoms based on `geom_dotsinterval()` create dotplots that automatically ensure the plot fits within the available space.

Roughly equivalent to:

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

Usage

```
geom_dots(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  binwidth = NA,
  dotsize = 1.07,
  stackratio = 1,
  layout = "bin",
  overlaps = "nudge",
  smooth = "none",
  overflow = "warn",
  verbose = FALSE,
  orientation = NA,
  subguide = "slab",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

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

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

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

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

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

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

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

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

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

`binwidth`
: <numeric | unit> The bin width to use for laying out the dots. One of:

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

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

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

`stackratio` <scalar numeric> The distance between the center of the dots in the same stack relative to the dot height. The default, `1`, makes dots in the same stack just touch each other.

`layout` <string> The layout method used for the dots. One of:

- `"bin"` (default): places dots on the off-axis at the midpoint of their bins as in the classic Wilkinson dotplot. This maintains the alignment of rows and columns in the dotplot. This layout is slightly different from the classic Wilkinson algorithm in that: (1) it nudges bins slightly to avoid overlapping bins and (2) if the input data are symmetrical it will return a symmetrical layout.
- `"weave"`: uses the same basic binning approach of `"bin"`, but places dots in the off-axis at their actual positions (unless `overlaps = "nudge"`, in which case overlaps may be nudged out of the way). This maintains the alignment of rows but does not align dots within columns.
- `"hex"`: uses the same basic binning approach of `"bin"`, but alternates placing dots `+ binwidth/4` or `- binwidth/4` in the off-axis from the bin center. This allows hexagonal packing by setting a `stackratio` less than 1 (something like `0.9` tends to work).
- `"swarm"`: uses the `"compactswarm"` layout from `beeswarm::beeswarm()`. Does not maintain alignment of rows or columns, but can be more compact and neat looking, especially for sample data (as opposed to quantile dotplots of theoretical distributions, which may look better with `"bin"`, `"weave"`, or `"hex"`).
- `"bar"`: for discrete distributions, lays out duplicate values in rectangular bars.

`overlaps` <string> How to handle overlapping dots or bins in the `"bin"`, `"weave"`, and `"hex"` layouts (dots never overlap in the `"swarm"` or `"bar"` layouts). For the purposes of this argument, dots are only considered to be overlapping if they would be overlapping when `dotsize = 1` and `stackratio = 1`; i.e. if you set those arguments to other values, overlaps may still occur. One of:

- `"keep"`: leave overlapping dots as they are. Dots may overlap (usually only slightly) in the `"bin"`, `"weave"`, and `"hex"` layouts.
- `"nudge"`: nudge overlapping dots out of the way. Overlaps are avoided using a constrained optimization which minimizes the squared distance of dots to their desired positions, subject to the constraint that adjacent dots do not overlap.

`smooth` <function | string> Smoother to apply to dot positions. One of:

- A function that takes a numeric vector of dot positions and returns a smoothed version of that vector, such as `smooth_bounded()`, `smooth_unbounded()`, smooth_discrete()`, or` smooth_bar()‘.
- A string indicating what smoother to use, as the suffix to a function name starting with `smooth_`; e.g. `"none"` (the default) applies `smooth_none()`, which simply returns the given vector without applying smoothing.

Smoothing is most effective when the smoother is matched to the support of the distribution; e.g. using `smooth_bounded(bounds = ...)`.

`overflow` <string> How to handle overflow of dots beyond the extent of the geom when a minimum `binwidth` (or an exact `binwidth`) is supplied. One of:

- `"keep"`: Keep the overflow, drawing dots outside the geom bounds.
- `"warn"`: Keep the overflow, but produce a warning suggesting solutions, such as setting `binwidth = NA` or `overflow = "compress"`.
- `"compress"`: Compress the layout. Reduces the `binwidth` to the size necessary to keep the dots within bounds, then adjusts `stackratio` and `dotsize` so that the apparent dot size is the user-specified minimum `binwidth` times the user-specified `dotsize`.

If you find the default layout has dots that are too small, and you are okay with dots overlapping, consider setting `overflow = "compress"` and supplying an exact or minimum dot size using `binwidth`.

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

`orientation` <string> Whether this geom is drawn horizontally or vertically. One of:

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

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

`subguide` <function | string> Sub-guide used to annotate the `thickness` scale. One of:

- A function that takes a `scale` argument giving a ggplot2::Scale object and an `orientation` argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as `subguide_axis()` (to draw a traditional axis) or `subguide_none()` (to draw no annotation). See `subguide_axis()` for a list of possibilities and examples.

- A string giving the name of such a function when prefixed with "subguide_"; e.g. "axis" or "none". The values "slab", "dots", and "spike" use the default subguide for their geom families (no subguide), which can be modified by setting subguide_slab, subguide_dots, or subguide_spike; see the documentation for those functions.

na.rm — <scalar logical> 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

The *dots* family of stats and geoms are similar to ggplot2::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 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)

Stats and geoms in this family include:

- geom_dots(): dotplots on raw data. Ensures the dotplot fits within available space by reducing the size of the dots automatically (may result in very small dots).
- geom_swarm() and geom_weave(): dotplots on raw data with defaults intended to create "beeswarm" plots. Used side = "both" by default, and sets the default dot size to the same size as geom_point() (binwidth = unit(1.5, "mm")), allowing dots to overlap instead of getting very small.
- stat_dots(): dotplots on raw data, **distributional** objects, and posterior::rvar()s
- geom_dotsinterval(): dotplot + interval plots on raw data with already-calculated intervals (rarely useful directly).
- stat_dotsinterval(): dotplot + interval plots on raw data, **distributional** objects, and posterior::rvar()s (will calculate intervals for you).
- geom_blur_dots(): blurry dotplots that allow the standard deviation of a blur applied to each dot to be specified using the sd aesthetic.

- stat_mcse_dots(): blurry dotplots of quantiles using the Monte Carlo Standard Error of each quantile.

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

Value

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

Aesthetics

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

Positional aesthetics

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

Dots-specific (aka Slab-specific) aesthetics

- family: The font family used to draw the dots.
- order: The order in which data points are stacked within bins. Can be used to create the effect of "stacked" dots by ordering dots according to a discrete variable. If omitted (NULL), the value of the data points themselves are used to determine stacking order. Only applies when layout is "bin" or "hex", as the other layout methods fully determine both *x* and *y* positions.
- side: Which side to place the slab on. "topright", "top", and "right" are synonyms which cause the slab to be drawn on the top or the right depending on if orientation is "horizontal" or "vertical". "bottomleft", "bottom", and "left" are synonyms which cause the slab to be drawn on the bottom or the left depending on if orientation is "horizontal" or "vertical". "topleft" causes the slab to be drawn on the top or the left, and "bottomright" causes the slab to be drawn on the bottom or the right. "both" draws the slab mirrored on both sides (as in a violin plot).
- scale: What proportion of the region allocated to this geom to use to draw the slab. If scale = 1, slabs that use the maximum range will just touch each other. Default is 0.9 to leave some space between adjacent slabs. For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.
- justification: Justification of the interval relative to the slab, where 0 indicates bottom/left justification and 1 indicates top/right justification (depending on orientation). If justification is NULL (the default), then it is set automatically based on the value of side: when side is "top"/"right" justification is set to 0, when side is "bottom"/"left" justification is set to 1, and when side is "both" justification is set to 0.5.
- datatype: When using composite geoms directly without a stat (e.g. geom_slabinterval()), datatype is used to indicate which part of the geom a row in the data targets: rows with datatype = "slab" target the slab portion of the geometry and rows with datatype = "interval" target the interval portion of the geometry. This is set automatically when using ggdist stats.

Interval-specific aesthetics

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

Point-specific aesthetics

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

Color aesthetics

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

Line aesthetics

- `linewidth`: Width of the line used to draw the **interval** (except with `geom_slab()`: then it is the width of the **slab**). With composite geometries including an interval and slab, use `slab_linewidth` to set the line width of the **slab** (see below). For **interval**, raw `linewidth` values are transformed according to the `interval_size_domain` and `interval_size_range` parameters of the `geom` (see above).
- `size`: Determines the size of the **point**. If `linewidth` is not provided, `size` will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only `size` and not `linewidth`). Raw `size` values are transformed according to the `interval_size_domain`, `interval_size_range`, and `fatten_point` parameters of the `geom` (see above). Use the `point_size` aesthetic (below) to set sub-geometry size directly without applying the effects of `interval_size_domain`, `interval_size_range`, and `fatten_point`.
- `stroke`: Width of the outline around the **point** sub-geometry.
- `linetype`: Type of line (e.g., `"solid"`, `"dashed"`, etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the `slab_linetype` or `interval_linetype` aesthetics (below) to set sub-geometry line types separately.

Slab-specific color and line override aesthetics

- `slab_fill`: Override for `fill`: the fill color of the slab.
- `slab_colour`: (or `slab_color`) Override for `colour`/`color`: the outline color of the slab.

- slab_alpha: Override for alpha: the opacity of the slab.
- slab_linewidth: Override for linwidth: the width of the outline of the slab.
- slab_linetype: Override for linetype: the line type of the outline of the slab.
- slab_shape: Override for shape: the shape of the dots used to draw the dotplot slab.

Interval-specific color and line override aesthetics

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

Point-specific color and line override aesthetics

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

Deprecated aesthetics

- slab_size: Use slab_linewidth.
- interval_size: Use interval_linewidth.

Other aesthetics (these work as in standard geoms)

- width
- height
- group

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

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_dots() for the stat version, intended for use on sample data or analytical distributions.

See geom_dotsinterval() for the geometry this shortcut is based on.

See vignette("dotsinterval") for a variety of examples of use.

Other dotsinterval geoms: geom_blur_dots(), geom_dotsinterval(), geom_swarm(), geom_weave()

Examples

```
library(dplyr)
library(ggplot2)

theme_set(theme_ggdist())

set.seed(12345)
df = tibble(
  g = rep(c("a", "b"), 200),
  value = rnorm(400, c(0, 3), c(0.75, 1))
)

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

df %>%
  ggplot(aes(x = value, y = g)) +
  geom_dots()

df %>%
  ggplot(aes(y = value, x = g)) +
  geom_dots()
```

---

geom_dotsinterval *Automatic dotplot + point + interval meta-geom*

Description

This meta-geom supports drawing combinations of dotplots, points, and intervals. Geoms and stats based on `geom_dotsinterval()` create dotplots that automatically determine a bin width that ensures the plot fits within the available space. They also ensure dots do not overlap, and allow the generation of quantile dotplots using the `quantiles` argument to `stat_dotsinterval()`/`stat_dots()`. Generally follows the naming scheme and arguments of the `geom_slabinterval()` and `stat_slabinterval()` family of geoms and stats.

Usage

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

```
  smooth = "none",
  overflow = "warn",
  verbose = FALSE,
  orientation = NA,
  interval_size_domain = c(1, 6),
  interval_size_range = c(0.6, 1.4),
  fatten_point = 1.8,
  arrow = NULL,
  show_slab = TRUE,
  show_point = TRUE,
  show_interval = TRUE,
  subguide = "slab",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

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

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

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

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

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

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

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

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

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

`binwidth` <numeric | unit> The bin width to use for laying out the dots. One of:

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

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

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

`stackratio` <scalar numeric> The distance between the center of the dots in the same stack relative to the dot height. The default, `1`, makes dots in the same stack just touch each other.

`layout` <string> The layout method used for the dots. One of:

- `"bin"` (default): places dots on the off-axis at the midpoint of their bins as in the classic Wilkinson dotplot. This maintains the alignment of rows and columns in the dotplot. This layout is slightly different from the classic Wilkinson algorithm in that: (1) it nudges bins slightly to avoid overlapping bins and (2) if the input data are symmetrical it will return a symmetrical layout.
- `"weave"`: uses the same basic binning approach of `"bin"`, but places dots in the off-axis at their actual positions (unless `overlaps = "nudge"`, in which case overlaps may be nudged out of the way). This maintains the alignment of rows but does not align dots within columns.
- `"hex"`: uses the same basic binning approach of `"bin"`, but alternates placing dots `+ binwidth/4` or `- binwidth/4` in the off-axis from the bin center. This allows hexagonal packing by setting a `stackratio` less than 1 (something like `0.9` tends to work).
- `"swarm"`: uses the `"compactswarm"` layout from `beeswarm::beeswarm()`. Does not maintain alignment of rows or columns, but can be more compact

and neat looking, especially for sample data (as opposed to quantile dotplots of theoretical distributions, which may look better with `"bin"`, `"weave"`, or `"hex"`).

- `"bar"`: for discrete distributions, lays out duplicate values in rectangular bars.

`overlaps` <string> How to handle overlapping dots or bins in the `"bin"`, `"weave"`, and `"hex"` layouts (dots never overlap in the `"swarm"` or `"bar"` layouts). For the purposes of this argument, dots are only considered to be overlapping if they would be overlapping when `dotsize = 1` and `stackratio = 1`; i.e. if you set those arguments to other values, overlaps may still occur. One of:

- `"keep"`: leave overlapping dots as they are. Dots may overlap (usually only slightly) in the `"bin"`, `"weave"`, and `"hex"` layouts.
- `"nudge"`: nudge overlapping dots out of the way. Overlaps are avoided using a constrained optimization which minimizes the squared distance of dots to their desired positions, subject to the constraint that adjacent dots do not overlap.

`smooth` <function | string> Smoother to apply to dot positions. One of:

- A function that takes a numeric vector of dot positions and returns a smoothed version of that vector, such as `smooth_bounded()`, `smooth_unbounded()`, smooth_discrete()`, or `smooth_bar()‘.
- A string indicating what smoother to use, as the suffix to a function name starting with `smooth_`; e.g. `"none"` (the default) applies `smooth_none()`, which simply returns the given vector without applying smoothing.

Smoothing is most effective when the smoother is matched to the support of the distribution; e.g. using `smooth_bounded(bounds = ...)`.

`overflow` <string> How to handle overflow of dots beyond the extent of the geom when a minimum `binwidth` (or an exact `binwidth`) is supplied. One of:

- `"keep"`: Keep the overflow, drawing dots outside the geom bounds.
- `"warn"`: Keep the overflow, but produce a warning suggesting solutions, such as setting `binwidth = NA` or `overflow = "compress"`.
- `"compress"`: Compress the layout. Reduces the `binwidth` to the size necessary to keep the dots within bounds, then adjusts `stackratio` and `dotsize` so that the apparent dot size is the user-specified minimum `binwidth` times the user-specified `dotsize`.

If you find the default layout has dots that are too small, and you are okay with dots overlapping, consider setting `overflow = "compress"` and supplying an exact or minimum dot size using `binwidth`.

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

`orientation` <string> Whether this geom is drawn horizontally or vertically. One of:

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

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

`interval_size_domain`
: <length-2 numeric> Minimum and maximum of the values of the `size` and `linewidth` aesthetics that will be translated into actual sizes for intervals drawn according to `interval_size_range` (see the documentation for that argument.)

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

`fatten_point`
: <scalar numeric> 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`.

`arrow`
: <arrow | NULL> Type of arrow heads to use on the interval, or `NULL` for no arrows.

`show_slab`
: <scalar logical> Should the slab portion of the geom be drawn?

`show_point`
: <scalar logical> Should the point portion of the geom be drawn?

`show_interval`
: <scalar logical> Should the interval portion of the geom be drawn?

`subguide`
: <function | string> Sub-guide used to annotate the `thickness` scale. One of:

- A function that takes a `scale` argument giving a ggplot2::Scale object and an `orientation` argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as `subguide_axis()` (to draw a traditional axis) or `subguide_none()` (to draw no annotation). See `subguide_axis()` for a list of possibilities and examples.

- A string giving the name of such a function when prefixed with "subguide_"; e.g. "axis" or "none". The values "slab", "dots", and "spike" use the default subguide for their geom families (no subguide), which can be modified by setting subguide_slab, subguide_dots, or subguide_spike; see the documentation for those functions.

na.rm <scalar logical> 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

The *dots* family of stats and geoms are similar to ggplot2::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 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)

Stats and geoms in this family include:

- geom_dots(): dotplots on raw data. Ensures the dotplot fits within available space by reducing the size of the dots automatically (may result in very small dots).
- geom_swarm() and geom_weave(): dotplots on raw data with defaults intended to create "beeswarm" plots. Used side = "both" by default, and sets the default dot size to the same size as geom_point() (binwidth = unit(1.5, "mm")), allowing dots to overlap instead of getting very small.
- stat_dots(): dotplots on raw data, **distributional** objects, and posterior::rvar()s
- geom_dotsinterval(): dotplot + interval plots on raw data with already-calculated intervals (rarely useful directly).
- stat_dotsinterval(): dotplot + interval plots on raw data, **distributional** objects, and posterior::rvar()s (will calculate intervals for you).
- geom_blur_dots(): blurry dotplots that allow the standard deviation of a blur applied to each dot to be specified using the sd aesthetic.

- stat_mcse_dots(): blurry dotplots of quantiles using the Monte Carlo Standard Error of each quantile.

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

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

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

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

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

Value

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

Aesthetics

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

Positional aesthetics

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

Dots-specific (aka Slab-specific) aesthetics

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

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

Interval-specific aesthetics

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

Point-specific aesthetics

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

Color aesthetics

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

Line aesthetics

- `linewidth`: Width of the line used to draw the **interval** (except with `geom_slab()`: then it is the width of the **slab**). With composite geometries including an interval and slab, use `slab_linewidth` to set the line width of the **slab** (see below). For **interval**, raw `linewidth` values are transformed according to the `interval_size_domain` and `interval_size_range` parameters of the `geom` (see above).
- `size`: Determines the size of the **point**. If `linewidth` is not provided, `size` will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only `size` and not `linewidth`). Raw `size` values are transformed according to the `interval_size_domain`, `interval_size_range`, and `fatten_point` parameters of the `geom` (see above). Use the `point_size` aesthetic (below) to set sub-geometry size directly without applying the effects of `interval_size_domain`, `interval_size_range`, and `fatten_point`.
- `stroke`: Width of the outline around the **point** sub-geometry.
- `linetype`: Type of line (e.g., `"solid"`, `"dashed"`, etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the `slab_linetype` or `interval_linetype` aesthetics (below) to set sub-geometry line types separately.

Slab-specific color and line override aesthetics

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

Interval-specific color and line override aesthetics

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

Point-specific color and line override aesthetics

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

Deprecated aesthetics

- `slab_size`: Use `slab_linewidth`.
- `interval_size`: Use `interval_linewidth`.

Other aesthetics (these work as in standard `geom`s)

- width
- height
- group

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

Author(s)

Matthew Kay

References

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

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

Other dotsinterval geoms: geom_blur_dots(), geom_dots(), geom_swarm(), geom_weave()

Examples

```
library(dplyr)
library(ggplot2)

theme_set(theme_ggdist())

set.seed(12345)
df = tibble(
  g = rep(c("a", "b"), 200),
  value = rnorm(400, c(0, 3), c(0.75, 1))
)


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

df %>%
  ggplot(aes(x = value, y = g)) +
  geom_dotsinterval()

df %>%
  ggplot(aes(y = value, x = g)) +
```

```
  geom_dotsinterval()


# stat_dots can summarize quantiles, creating quantile dotplots

data(RankCorr_u_tau, package = "ggdist")

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 = after_stat(x > 6))) +
  stat_dotsinterval(quantiles = 100)
```

`geom_interval` *Multiple-interval plot (shortcut geom)*

Description

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

Roughly equivalent to:

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

Usage

```
geom_interval(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  orientation = NA,
  interval_size_range = c(1, 6),
  interval_size_domain = c(1, 6),
```

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

Arguments

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

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

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

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

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

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

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

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

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

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

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

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

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

`interval_size_domain`
: <length-2 numeric> Minimum and maximum of the values of the `size` and `linewidth` aesthetics that will be translated into actual sizes for intervals drawn according to `interval_size_range` (see the documentation for that argument.)

`arrow`
: <arrow | NULL> Type of arrow heads to use on the interval, or `NULL` for no arrows.

`na.rm`
: <scalar logical> If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

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

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

Details

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

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

Value

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

Aesthetics

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

Positional aesthetics

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

Interval-specific aesthetics

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

Color aesthetics

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

Line aesthetics

- `linewidth`: Width of the line used to draw the **interval** (except with `geom_slab()`: then it is the width of the **slab**). With composite geometries including an interval and slab, use `slab_linewidth` to set the line width of the **slab** (see below). For **interval**, raw `linewidth` values are transformed according to the `interval_size_domain` and `interval_size_range` parameters of the `geom` (see above).
- `size`: Determines the size of the **point**. If `linewidth` is not provided, `size` will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only `size` and not `linewidth`). Raw `size` values are transformed according to the `interval_size_domain`, `interval_size_range`, and `fatten_point` parameters of the `geom` (see above). Use the `point_size` aesthetic (below) to set sub-geometry size directly without applying the effects of `interval_size_domain`, `interval_size_range`, and `fatten_point`.
- `stroke`: Width of the outline around the **point** sub-geometry.
- `linetype`: Type of line (e.g., `"solid"`, `"dashed"`, etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the `slab_linetype` or `interval_linetype` aesthetics (below) to set sub-geometry line types separately.

Interval-specific color and line override aesthetics

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

Deprecated aesthetics

- interval_size: Use interval_linewidth.

Other aesthetics (these work as in standard geoms)

- width
- height
- group

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

See Also

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

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

Examples

```
library(dplyr)
library(ggplot2)

theme_set(theme_ggdist())

data(RankCorr_u_tau, package = "ggdist")

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

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

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

```
geom_interval() +
scale_color_brewer()
```

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

Description

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

Usage

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

Arguments

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

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

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

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

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

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

  - A Stat ggproto subclass, for example StatCount.

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

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

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

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

`step` <scalar logical | string> Should the line/ribbon be drawn as a step function? One of:

- `FALSE` (default): do not draw as a step function.
- `"mid"` (or `TRUE`): draw steps midway between adjacent x values.
- `"hv"`: draw horizontal-then-vertical steps.
- `"vh"`: draw as vertical-then-horizontal steps.

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

`orientation` <string> Whether this geom is drawn horizontally or vertically. One of:

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

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

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

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

Value

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

Aesthetics

The line+ribbon stats and geoms have a wide variety of aesthetics that control the appearance of their two sub-geometries: the **line** and the **ribbon**.

Positional aesthetics

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

Ribbon-specific aesthetics

- xmin: Left edge of the ribbon sub-geometry (if orientation = "horizontal").
- xmax: Right edge of the ribbon sub-geometry (if orientation = "horizontal").
- ymin: Lower edge of the ribbon sub-geometry (if orientation = "vertical").
- ymax: Upper edge of the ribbon sub-geometry (if orientation = "vertical").
- order: The order in which ribbons are drawn. Ribbons with the smallest mean value of order are drawn first (i.e., will be drawn below ribbons with larger mean values of order). If order is not supplied to geom_lineribbon(), -abs(xmax - xmin) or -abs(ymax - ymax) (depending on orientation) is used, having the effect of drawing the widest (on average) ribbons on the bottom. stat_lineribbon() uses order = after_stat(level) by default, causing the ribbons generated from the largest .width to be drawn on the bottom.

Color aesthetics

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

Line aesthetics

- `linewidth`: Width of **line**. In **ggplot2** < 3.4, was called `size`.
- `linetype`: Type of **line** (e.g., `"solid"`, `"dashed"`, etc)

Other aesthetics (these work as in standard geoms)

- `group`

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

Author(s)

Matthew Kay

See Also

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

Examples

```
library(dplyr)
library(ggplot2)

theme_set(theme_ggdist())

set.seed(12345)
tibble(
  x = rep(1:10, 100),
  y = rnorm(1000, x)
) %>%
  group_by(x) %>%
  median_qi(.width = c(.5, .8, .95)) %>%
  ggplot(aes(x = x, y = y, ymin = .lower, ymax = .upper)) +
  # automatically uses aes(fill = forcats::fct_rev(ordered(.width)))
  geom_lineribbon() +
  scale_fill_brewer()
```

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

Description

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

Roughly equivalent to:

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

Usage

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

Arguments

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

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

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

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

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

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

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

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

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

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

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

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

interval_size_domain
: <length-2 numeric> Minimum and maximum of the values of the size and linewidth aesthetics that will be translated into actual sizes for intervals drawn according to interval_size_range (see the documentation for that argument.)

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

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

`fatten_point` <scalar numeric> 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`.

`arrow` <arrow | NULL> Type of arrow heads to use on the interval, or `NULL` for no arrows.

`na.rm` <scalar logical> 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? 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). It can also be a named logical vector to finely select the aesthetics to display.

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

Details

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

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

Value

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

Aesthetics

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

Positional aesthetics

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

Interval-specific aesthetics

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

Point-specific aesthetics

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

Color aesthetics

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

Line aesthetics

- `linewidth`: Width of the line used to draw the **interval** (except with `geom_slab()`: then it is the width of the **slab**). With composite geometries including an interval and slab, use `slab_linewidth` to set the line width of the **slab** (see below). For **interval**, raw `linewidth` values are transformed according to the `interval_size_domain` and `interval_size_range` parameters of the `geom` (see above).
- `size`: Determines the size of the **point**. If `linewidth` is not provided, `size` will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only `size` and not `linewidth`). Raw `size` values are transformed according to the `interval_size_domain`, `interval_size_range`, and `fatten_point` parameters of the `geom` (see above). Use the `point_size` aesthetic (below) to set sub-geometry size directly without applying the effects of `interval_size_domain`, `interval_size_range`, and `fatten_point`.
- `stroke`: Width of the outline around the **point** sub-geometry.
- `linetype`: Type of line (e.g., `"solid"`, `"dashed"`, etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the `slab_linetype` or `interval_linetype` aesthetics (below) to set sub-geometry line types separately.

Interval-specific color and line override aesthetics

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

- interval_alpha: Override for alpha: the opacity of the interval.
- interval_linetype: Override for linetype: the line type of the interval.

Point-specific color and line override aesthetics

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

Deprecated aesthetics

- interval_size: Use interval_linewidth.

Other aesthetics (these work as in standard geoms)

- width
- height
- group

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

See Also

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

Other slabinterval geoms: geom_interval(), geom_slab(), geom_spike()

Examples

```
library(dplyr)
library(ggplot2)

data(RankCorr_u_tau, package = "ggdist")

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

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

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

geom_slab *Slab (ridge) plot (shortcut geom)*

Description

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

Roughly equivalent to:

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

Usage

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

Arguments

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

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

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

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

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

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

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

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

subscale
: <function | string> Sub-scale used to scale values of the `thickness` aesthetic within the groups determined by `normalize`. One of:

  - A function that takes an `x` argument giving a numeric vector of values to be scaled and then returns a thickness vector representing the scaled values, such as `subscale_thickness()` or `subscale_identity()`.
  - A string giving the name of such a function when prefixed with `"subscale_"`; e.g. `"thickness"` or `"identity"`. The value `"thickness"` using the default subscale, which can be modified by setting `subscale_thickness`; see the documentation for that function.

  For a comprehensive discussion and examples of slab scaling and normalization, see the `thickness` scale article.

normalize
: <string> Groups within which to scale values of the `thickness` aesthetic. One of:

  - `"all"`: normalize so that the maximum height across all data is `1`.

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

For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.

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

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

subguide
<function | string> Sub-guide used to annotate the thickness scale. One of:

- A function that takes a scale argument giving a ggplot2::Scale object and an orientation argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as subguide_axis() (to draw a traditional axis) or subguide_none() (to draw no annotation). See subguide_axis() for a list of possibilities and examples.
- A string giving the name of such a function when prefixed with "subguide_"; e.g. "axis" or "none". The values "slab", "dots", and "spike" use the default subguide for their geom families (no subguide), which can be modified by setting subguide_slab, subguide_dots, or subguide_spike; see the documentation for those functions.

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

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

Value

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

Aesthetics

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

Positional aesthetics

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

Slab-specific aesthetics

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

Color aesthetics

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

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

Line aesthetics

- `linewidth`: Width of the line used to draw the **interval** (except with `geom_slab()`: then it is the width of the **slab**). With composite geometries including an interval and slab, use `slab_linewidth` to set the line width of the **slab** (see below). For **interval**, raw `linewidth` values are transformed according to the `interval_size_domain` and `interval_size_range` parameters of the `geom` (see above).
- `size`: Determines the size of the **point**. If `linewidth` is not provided, `size` will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only `size` and not `linewidth`). Raw `size` values are transformed according to the `interval_size_domain`, `interval_size_range`, and `fatten_point` parameters of the `geom` (see above). Use the `point_size` aesthetic (below) to set sub-geometry size directly without applying the effects of `interval_size_domain`, `interval_size_range`, and `fatten_point`.
- `stroke`: Width of the outline around the **point** sub-geometry.
- `linetype`: Type of line (e.g., `"solid"`, `"dashed"`, etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the `slab_linetype` or `interval_linetype` aesthetics (below) to set sub-geometry line types separately.

Slab-specific color and line override aesthetics

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

Deprecated aesthetics

- `slab_size`: Use `slab_linewidth`.

Other aesthetics (these work as in standard geoms)

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

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

See Also

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

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

Examples

```
library(dplyr)
library(ggplot2)

theme_set(theme_ggdist())

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

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

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

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

---

geom_slabinterval *Slab + point + interval meta-geom*

Description

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

Usage

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

Arguments

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

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

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

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

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

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

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

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

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

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

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

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

subscale
: <function | string> Sub-scale used to scale values of the `thickness` aesthetic within the groups determined by `normalize`. One of:

- A function that takes an `x` argument giving a numeric vector of values to be scaled and then returns a thickness vector representing the scaled values, such as `subscale_thickness()` or `subscale_identity()`.
- A string giving the name of such a function when prefixed with `"subscale_"`; e.g. `"thickness"` or `"identity"`. The value `"thickness"` using the default subscale, which can be modified by setting `subscale_thickness`; see the documentation for that function.

For a comprehensive discussion and examples of slab scaling and normalization, see the `thickness` scale article.

normalize
: <string> Groups within which to scale values of the `thickness` aesthetic. One of:

- `"all"`: normalize so that the maximum height across all data is `1`.

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

For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.

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

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

interval_size_domain
: <length-2 numeric> Minimum and maximum of the values of the size and linewidth aesthetics that will be translated into actual sizes for intervals drawn according to interval_size_range (see the documentation for that argument.)

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

targeting the point and interval separately. If you want to adjust the size of the interval or points separately, you can also use the `linewidth` or `point_size` aesthetics; see sub-geometry-scales.

`fatten_point` <scalar numeric> 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`.

`arrow` <arrow | NULL> Type of arrow heads to use on the interval, or `NULL` for no arrows.

`show_slab` <scalar logical> Should the slab portion of the geom be drawn?

`show_point` <scalar logical> Should the point portion of the geom be drawn?

`show_interval` <scalar logical> Should the interval portion of the geom be drawn?

`subguide` <function | string> Sub-guide used to annotate the `thickness` scale. One of:

- A function that takes a `scale` argument giving a ggplot2::Scale object and an `orientation` argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as `subguide_axis()` (to draw a traditional axis) or `subguide_none()` (to draw no annotation). See `subguide_axis()` for a list of possibilities and examples.
- A string giving the name of such a function when prefixed with `"subguide_"`; e.g. `"axis"` or `"none"`. The values `"slab"`, `"dots"`, and `"spike"` use the default subguide for their geom families (no subguide), which can be modified by setting `subguide_slab`, `subguide_dots`, or `subguide_spike`; see the documentation for those functions.

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

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

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

Details

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

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

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

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

Wrapper geoms include:

- geom_pointinterval()
- geom_interval()
- geom_slab()

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

Value

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

Aesthetics

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

Positional aesthetics

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

Slab-specific aesthetics

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

- `scale`: What proportion of the region allocated to this geom to use to draw the slab. If `scale = 1`, slabs that use the maximum range will just touch each other. Default is `0.9` to leave some space between adjacent slabs. For a comprehensive discussion and examples of slab scaling and normalization, see the `thickness` scale article.
- `justification`: Justification of the interval relative to the slab, where `0` indicates bottom/left justification and `1` indicates top/right justification (depending on `orientation`). If `justification` is `NULL` (the default), then it is set automatically based on the value of `side`: when `side` is `"top"/"right" justification` is set to `0`, when `side` is `"bottom"/"left" justification` is set to `1`, and when `side` is `"both" justification` is set to 0.5.
- `datatype`: When using composite geoms directly without a `stat` (e.g. `geom_slabinterval()`), `datatype` is used to indicate which part of the geom a row in the data targets: rows with `datatype = "slab"` target the slab portion of the geometry and rows with `datatype = "interval"` target the interval portion of the geometry. This is set automatically when using ggdist `stats`.

Interval-specific aesthetics

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

Point-specific aesthetics

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

Color aesthetics

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

Line aesthetics

- `linewidth`: Width of the line used to draw the **interval** (except with `geom_slab()`: then it is the width of the **slab**). With composite geometries including an interval and slab, use `slab_linewidth` to set the line width of the **slab** (see below). For **interval**, raw `linewidth` values are transformed according to the `interval_size_domain` and `interval_size_range` parameters of the `geom` (see above).

- `size`: Determines the size of the **point**. If `linewidth` is not provided, `size` will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only `size` and not `linewidth`). Raw `size` values are transformed according to the `interval_size_domain`, `interval_size_range`, and `fatten_point` parameters of the `geom` (see above). Use the `point_size` aesthetic (below) to set sub-geometry size directly without applying the effects of `interval_size_domain`, `interval_size_range`, and `fatten_point`.
- `stroke`: Width of the outline around the **point** sub-geometry.
- `linetype`: Type of line (e.g., `"solid"`, `"dashed"`, etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the `slab_linetype` or `interval_linetype` aesthetics (below) to set sub-geometry line types separately.

Slab-specific color and line override aesthetics

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

Interval-specific color and line override aesthetics

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

Point-specific color and line override aesthetics

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

Deprecated aesthetics

- `slab_size`: Use `slab_linewidth`.
- `interval_size`: Use `interval_linewidth`.

Other aesthetics (these work as in standard `geom`s)

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

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

Author(s)

Matthew Kay

See Also

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

Examples

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

---

geom_spike *Spike plot (ggplot2 geom)*

Description

Geometry for drawing "spikes" (optionally with points on them) on top of `geom_slabinterval()` geometries: this geometry understands the scaling and positioning of the `thickness` aesthetic from `geom_slabinterval()`, which allows you to position spikes and points along a slab.

Usage

```
geom_spike(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  subguide = "spike",
  orientation = NA,
  subscale = "thickness",
  normalize = "all",
  arrow = NULL,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

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

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

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

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

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

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

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

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

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

`subguide`
: <function | string> Sub-guide used to annotate the `thickness` scale. One of:

  - A function that takes a `scale` argument giving a ggplot2::Scale object and an `orientation` argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as `subguide_axis()` (to draw a traditional axis) or `subguide_none()` (to draw no annotation). See `subguide_axis()` for a list of possibilities and examples.
  - A string giving the name of such a function when prefixed with `"subguide_"`; e.g. `"axis"` or `"none"`. The values `"slab"`, `"dots"`, and `"spike"` use the default subguide for their geom families (no subguide), which can be modified by setting `subguide_slab`, `subguide_dots`, or `subguide_spike`; see the documentation for those functions.

`orientation`
: <string> Whether this geom is drawn horizontally or vertically. One of:

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

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

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

subscale
<function | string> Sub-scale used to scale values of the thickness aesthetic within the groups determined by normalize. One of:

- A function that takes an x argument giving a numeric vector of values to be scaled and then returns a thickness vector representing the scaled values, such as subscale_thickness() or subscale_identity().
- A string giving the name of such a function when prefixed with "subscale_"; e.g. "thickness" or "identity". The value "thickness" using the default subscale, which can be modified by setting subscale_thickness; see the documentation for that function.

For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.

normalize
<string> Groups within which to scale values of the thickness aesthetic. One of:

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

For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.

arrow
<arrow | NULL> Type of arrow heads to use on the spike, or NULL for no arrows.

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

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

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

Details

This geometry consists of a "spike" (vertical/horizontal line segment) and a "point" (at the end of the line segment). It uses the `thickness` aesthetic to determine where the endpoint of the line is, which allows it to be used with `geom_slabinterval()` geometries for labeling specific values of the thickness function.

Value

A ggplot2::Geom representing a spike geometry which can be added to a `ggplot()` object. rd_slabinterval_aesthetics(geom_r

Aesthetics

The spike `geom` has a wide variety of aesthetics that control the appearance of its two sub-geometries: the **spike** and the **point**.

Positional aesthetics

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

Spike-specific (aka Slab-specific) aesthetics

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

Color aesthetics

- `colour`: (or `color`) The color of the **spike** and **point** sub-geometries.
- `fill`: The fill color of the **point** sub-geometry.
- `alpha`: The opacity of the **spike** and **point** sub-geometries.
- `colour_ramp`: (or `color_ramp`) A secondary scale that modifies the `color` scale to "ramp" to another color. See `scale_colour_ramp()` for examples.
- `fill_ramp`: A secondary scale that modifies the `fill` scale to "ramp" to another color. See `scale_fill_ramp()` for examples.

Line aesthetics

- `linewidth`: Width of the line used to draw the **spike** sub-geometry.

- size: Size of the **point** sub-geometry.
- stroke: Width of the outline around the **point** sub-geometry.
- linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the **spike**.

Other aesthetics (these work as in standard geoms)

- width
- height
- group

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

See Also

See stat_spike() for the stat version, intended for use on sample data or analytical distributions.

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

Examples

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

# geom_spike is easiest to use with distributional or
# posterior::rvar objects
df = tibble(
  d = dist_normal(1:2, 1:2), g = c("a", "b")
)

# annotate the density at the mean of a distribution
df %>% mutate(
  mean = mean(d),
  density(d, list(density_at_mean = mean))
) %>%
  ggplot(aes(y = g)) +
  stat_slab(aes(xdist = d)) +
  geom_spike(aes(x = mean, thickness = density_at_mean)) +
  # need shared thickness scale so that stat_slab and geom_spike line up
  scale_thickness_shared()

# annotate the endpoints of intervals of a distribution
# here we'll use an arrow instead of a point by setting size = 0
arrow_spec = arrow(angle = 45, type = "closed", length = unit(4, "pt"))
df %>% mutate(
  median_qi(d, .width = 0.9),
  density(d, list(density_lower = .lower, density_upper = .upper))
) %>%
  ggplot(aes(y = g)) +
```

```
  stat_halfeye(aes(xdist = d), .width = 0.9, color = "gray35") +
  geom_spike(
    aes(x = .lower, thickness = density_lower),
    size = 0, arrow = arrow_spec, color = "blue", linewidth = 0.75
  ) +
  geom_spike(
    aes(x = .upper, thickness = density_upper),
    size = 0, arrow = arrow_spec, color = "red", linewidth = 0.75
  ) +
  scale_thickness_shared()
```

`geom_swarm` *Beeswarm plot (shortcut geom)*

Description

Shortcut version of `geom_dotsinterval()` for creating beeswarm plots. Geoms based on `geom_dotsinterval()` create dotplots that automatically ensure the plot fits within the available space.

Roughly equivalent to:

```
geom_dots(
  aes(side = "both"),
  overflow = "compress",
  binwidth = unit(1.5, "mm"),
  layout = "swarm"
)
```

Usage

```
geom_swarm(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  overflow = "compress",
  binwidth = unit(1.5, "mm"),
  layout = "swarm",
  dotsize = 1.07,
  stackratio = 1,
  overlaps = "nudge",
  smooth = "none",
  verbose = FALSE,
  orientation = NA,
  subguide = "slab",
  na.rm = FALSE,
```

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

Arguments

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

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

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

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

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

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

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

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

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

`overflow` <string> How to handle overflow of dots beyond the extent of the geom when a minimum `binwidth` (or an exact `binwidth`) is supplied. One of:

- `"keep"`: Keep the overflow, drawing dots outside the geom bounds.
- `"warn"`: Keep the overflow, but produce a warning suggesting solutions, such as setting `binwidth = NA` or `overflow = "compress"`.
- `"compress"`: Compress the layout. Reduces the `binwidth` to the size necessary to keep the dots within bounds, then adjusts `stackratio` and `dotsize` so that the apparent dot size is the user-specified minimum `binwidth` times the user-specified `dotsize`.

If you find the default layout has dots that are too small, and you are okay with dots overlapping, consider setting `overflow = "compress"` and supplying an exact or minimum dot size using `binwidth`.

`binwidth` <numeric | unit> The bin width to use for laying out the dots. One of:

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

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

`layout` <string> The layout method used for the dots. One of:

- `"bin"` (default): places dots on the off-axis at the midpoint of their bins as in the classic Wilkinson dotplot. This maintains the alignment of rows and columns in the dotplot. This layout is slightly different from the classic Wilkinson algorithm in that: (1) it nudges bins slightly to avoid overlapping bins and (2) if the input data are symmetrical it will return a symmetrical layout.
- `"weave"`: uses the same basic binning approach of `"bin"`, but places dots in the off-axis at their actual positions (unless `overlaps = "nudge"`, in which case overlaps may be nudged out of the way). This maintains the alignment of rows but does not align dots within columns.
- `"hex"`: uses the same basic binning approach of `"bin"`, but alternates placing dots `+ binwidth/4` or `- binwidth/4` in the off-axis from the bin center. This allows hexagonal packing by setting a `stackratio` less than 1 (something like `0.9` tends to work).
- `"swarm"`: uses the `"compactswarm"` layout from `beeswarm::beeswarm()`. Does not maintain alignment of rows or columns, but can be more compact and neat looking, especially for sample data (as opposed to quantile dotplots of theoretical distributions, which may look better with `"bin"`, `"weave"`, or `"hex"`).
- `"bar"`: for discrete distributions, lays out duplicate values in rectangular bars.

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

stackratio
: <scalar numeric> The distance between the center of the dots in the same stack relative to the dot height. The default, 1, makes dots in the same stack just touch each other.

overlaps
: <string> How to handle overlapping dots or bins in the "bin", "weave", and "hex" layouts (dots never overlap in the "swarm" or "bar" layouts). For the purposes of this argument, dots are only considered to be overlapping if they would be overlapping when dotsize = 1 and stackratio = 1; i.e. if you set those arguments to other values, overlaps may still occur. One of:

- "keep": leave overlapping dots as they are. Dots may overlap (usually only slightly) in the "bin", "weave", and "hex" layouts.
- "nudge": nudge overlapping dots out of the way. Overlaps are avoided using a constrained optimization which minimizes the squared distance of dots to their desired positions, subject to the constraint that adjacent dots do not overlap.

smooth
: <function | string> Smoother to apply to dot positions. One of:

- A function that takes a numeric vector of dot positions and returns a smoothed version of that vector, such as smooth_bounded(), smooth_unbounded(), smooth_discrete(), or smooth_bar()‘.
- A string indicating what smoother to use, as the suffix to a function name starting with smooth_; e.g. "none" (the default) applies smooth_none(), which simply returns the given vector without applying smoothing.

Smoothing is most effective when the smoother is matched to the support of the distribution; e.g. using smooth_bounded(bounds = ...).

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

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

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

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

subguide
: <function | string> Sub-guide used to annotate the thickness scale. One of:
  - A function that takes a scale argument giving a ggplot2::Scale object and an orientation argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as subguide_axis() (to draw a traditional axis) or subguide_none() (to draw no annotation). See subguide_axis() for a list of possibilities and examples.
  - A string giving the name of such a function when prefixed with "subguide_"; e.g. "axis" or "none". The values "slab", "dots", and "spike" use the default subguide for their geom families (no subguide), which can be modified by setting subguide_slab, subguide_dots, or subguide_spike; see the documentation for those functions.

na.rm
: <scalar logical> 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

The *dots* family of stats and geoms are similar to ggplot2::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 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)

Stats and geoms in this family include:

- geom_dots(): dotplots on raw data. Ensures the dotplot fits within available space by reducing the size of the dots automatically (may result in very small dots).
- geom_swarm() and geom_weave(): dotplots on raw data with defaults intended to create "beeswarm" plots. Used side = "both" by default, and sets the default dot size to the same size as geom_point() (binwidth = unit(1.5, "mm")), allowing dots to overlap instead of getting very small.
- stat_dots(): dotplots on raw data, **distributional** objects, and posterior::rvar()s
- geom_dotsinterval(): dotplot + interval plots on raw data with already-calculated intervals (rarely useful directly).

- `stat_dotsinterval()`: dotplot + interval plots on raw data, **distributional** objects, and `posterior::rvar()`s (will calculate intervals for you).
- `geom_blur_dots()`: blurry dotplots that allow the standard deviation of a blur applied to each dot to be specified using the `sd` aesthetic.
- `stat_mcse_dots()`: blurry dotplots of quantiles using the Monte Carlo Standard Error of each quantile.

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

Value

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

Aesthetics

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

Positional aesthetics

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

Dots-specific (aka Slab-specific) aesthetics

- `family`: The font family used to draw the dots.
- `order`: The order in which data points are stacked within bins. Can be used to create the effect of "stacked" dots by ordering dots according to a discrete variable. If omitted (`NULL`), the value of the data points themselves are used to determine stacking order. Only applies when `layout` is `"bin"` or `"hex"`, as the other layout methods fully determine both *x* and *y* positions.
- `side`: Which side to place the slab on. `"topright"`, `"top"`, and `"right"` are synonyms which cause the slab to be drawn on the top or the right depending on if `orientation` is `"horizontal"` or `"vertical"`. `"bottomleft"`, `"bottom"`, and `"left"` are synonyms which cause the slab to be drawn on the bottom or the left depending on if `orientation` is `"horizontal"` or `"vertical"`. `"topleft"` causes the slab to be drawn on the top or the left, and `"bottomright"` causes the slab to be drawn on the bottom or the right. `"both"` draws the slab mirrored on both sides (as in a violin plot).
- `scale`: What proportion of the region allocated to this geom to use to draw the slab. If `scale = 1`, slabs that use the maximum range will just touch each other. Default is `0.9` to leave some space between adjacent slabs. For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.
- `justification`: Justification of the interval relative to the slab, where `0` indicates bottom/left justification and `1` indicates top/right justification (depending on `orientation`). If `justification` is `NULL` (the default), then it is set automatically based on the value of `side`: when `side` is `"top"`/`"right"` `justification` is set to `0`, when `side` is `"bottom"`/`"left"` `justification` is set to `1`, and when `side` is `"both"` `justification` is set to 0.5.

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

Interval-specific aesthetics

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

Point-specific aesthetics

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

Color aesthetics

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

Line aesthetics

- `linewidth`: Width of the line used to draw the **interval** (except with `geom_slab()`: then it is the width of the **slab**). With composite geometries including an interval and slab, use `slab_linewidth` to set the line width of the **slab** (see below). For **interval**, raw `linewidth` values are transformed according to the `interval_size_domain` and `interval_size_range` parameters of the `geom` (see above).
- `size`: Determines the size of the **point**. If `linewidth` is not provided, `size` will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only `size` and not `linewidth`). Raw `size` values are transformed according to the `interval_size_domain`, `interval_size_range`, and `fatten_point` parameters of the `geom` (see above). Use the `point_size` aesthetic (below) to set sub-geometry size directly without applying the effects of `interval_size_domain`, `interval_size_range`, and `fatten_point`.
- `stroke`: Width of the outline around the **point** sub-geometry.
- `linetype`: Type of line (e.g., `"solid"`, `"dashed"`, etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the `slab_linetype` or `interval_linetype` aesthetics (below) to set sub-geometry line types separately.

Slab-specific color and line override aesthetics

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

Interval-specific color and line override aesthetics

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

Point-specific color and line override aesthetics

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

Deprecated aesthetics

- `slab_size`: Use `slab_linewidth`.
- `interval_size`: Use `interval_linewidth`.

Other aesthetics (these work as in standard `geom`s)

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

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

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 `geom_dotsinterval()` for the geometry this shortcut is based on.

See `vignette("dotsinterval")` for a variety of examples of use.

Other dotsinterval geoms: `geom_blur_dots()`, `geom_dots()`, `geom_dotsinterval()`, `geom_weave()`

Examples

```
library(dplyr)
library(ggplot2)

theme_set(theme_ggdist())

set.seed(12345)
df = tibble(
  g = rep(c("a", "b"), 200),
  value = rnorm(400, c(0, 3), c(0.75, 1))
)

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

df %>%
  ggplot(aes(x = value, y = g)) +
  geom_swarm()

df %>%
  ggplot(aes(y = value, x = g)) +
  geom_swarm()
```

`geom_weave` *Dot-weave plot (shortcut geom)*

Description

Shortcut version of `geom_dotsinterval()` for creating dot-weave plots. Geoms based on `geom_dotsinterval()` create dotplots that automatically ensure the plot fits within the available space.

Roughly equivalent to:

```
geom_dots(
  aes(side = "both"),
  layout = "weave",
  overflow = "compress",
  binwidth = unit(1.5, "mm")
)
```

Usage

```
geom_weave(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  layout = "weave",
  overflow = "compress",
  binwidth = unit(1.5, "mm"),
  dotsize = 1.07,
  stackratio = 1,
  overlaps = "nudge",
  smooth = "none",
  verbose = FALSE,
  orientation = NA,
  subguide = "slab",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

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

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

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

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

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

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

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

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

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

`layout` <string> The layout method used for the dots. One of:

- `"bin"` (default): places dots on the off-axis at the midpoint of their bins as in the classic Wilkinson dotplot. This maintains the alignment of rows and columns in the dotplot. This layout is slightly different from the classic Wilkinson algorithm in that: (1) it nudges bins slightly to avoid overlapping bins and (2) if the input data are symmetrical it will return a symmetrical layout.
- `"weave"`: uses the same basic binning approach of `"bin"`, but places dots in the off-axis at their actual positions (unless `overlaps = "nudge"`, in which case overlaps may be nudged out of the way). This maintains the alignment of rows but does not align dots within columns.
- `"hex"`: uses the same basic binning approach of `"bin"`, but alternates placing dots `+ binwidth/4` or `- binwidth/4` in the off-axis from the bin center. This allows hexagonal packing by setting a `stackratio` less than 1 (something like `0.9` tends to work).
- `"swarm"`: uses the `"compactswarm"` layout from `beeswarm::beeswarm()`. Does not maintain alignment of rows or columns, but can be more compact and neat looking, especially for sample data (as opposed to quantile dotplots of theoretical distributions, which may look better with `"bin"`, `"weave"`, or `"hex"`).
- `"bar"`: for discrete distributions, lays out duplicate values in rectangular bars.

`overflow` <string> How to handle overflow of dots beyond the extent of the geom when a minimum `binwidth` (or an exact `binwidth`) is supplied. One of:

- `"keep"`: Keep the overflow, drawing dots outside the geom bounds.
- `"warn"`: Keep the overflow, but produce a warning suggesting solutions, such as setting `binwidth = NA` or `overflow = "compress"`.
- `"compress"`: Compress the layout. Reduces the `binwidth` to the size necessary to keep the dots within bounds, then adjusts `stackratio` and `dotsize` so that the apparent dot size is the user-specified minimum `binwidth` times the user-specified `dotsize`.

If you find the default layout has dots that are too small, and you are okay with dots overlapping, consider setting `overflow = "compress"` and supplying an exact or minimum dot size using `binwidth`.

`binwidth` <numeric | unit> The bin width to use for laying out the dots. One of:

- `NA` (the default): Dynamically select the bin width based on the size of the plot when drawn. This will pick a `binwidth` such that the tallest stack of dots is at most `scale` in height (ideally exactly `scale` in height, though this is not guaranteed).

- A length-1 (scalar) numeric or unit object giving the exact bin width.
- A length-2 (vector) numeric or unit object giving the minimum and maximum desired bin width. The bin width will be dynamically selected within these bounds.

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

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

`stackratio` <scalar numeric> The distance between the center of the dots in the same stack relative to the dot height. The default, `1`, makes dots in the same stack just touch each other.

`overlaps` <string> How to handle overlapping dots or bins in the `"bin"`, `"weave"`, and `"hex"` layouts (dots never overlap in the `"swarm"` or `"bar"` layouts). For the purposes of this argument, dots are only considered to be overlapping if they would be overlapping when `dotsize = 1` and `stackratio = 1`; i.e. if you set those arguments to other values, overlaps may still occur. One of:

- `"keep"`: leave overlapping dots as they are. Dots may overlap (usually only slightly) in the `"bin"`, `"weave"`, and `"hex"` layouts.
- `"nudge"`: nudge overlapping dots out of the way. Overlaps are avoided using a constrained optimization which minimizes the squared distance of dots to their desired positions, subject to the constraint that adjacent dots do not overlap.

`smooth` <function | string> Smoother to apply to dot positions. One of:

- A function that takes a numeric vector of dot positions and returns a smoothed version of that vector, such as `smooth_bounded()`, `smooth_unbounded()`, smooth_discrete()`, or `smooth_bar()‘.
- A string indicating what smoother to use, as the suffix to a function name starting with `smooth_`; e.g. `"none"` (the default) applies `smooth_none()`, which simply returns the given vector without applying smoothing.

Smoothing is most effective when the smoother is matched to the support of the distribution; e.g. using `smooth_bounded(bounds = ...)`.

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

`orientation` <string> Whether this geom is drawn horizontally or vertically. One of:

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

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

`subguide` <function | string> Sub-guide used to annotate the `thickness` scale. One of:

- A function that takes a `scale` argument giving a ggplot2::Scale object and an `orientation` argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as `subguide_axis()` (to draw a traditional axis) or `subguide_none()` (to draw no annotation). See `subguide_axis()` for a list of possibilities and examples.
- A string giving the name of such a function when prefixed with `"subguide_"`; e.g. `"axis"` or `"none"`. The values `"slab"`, `"dots"`, and `"spike"` use the default subguide for their geom families (no subguide), which can be modified by setting `subguide_slab`, `subguide_dots`, or `subguide_spike`; see the documentation for those functions.

`na.rm` <scalar logical> 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

The *dots* family of stats and geoms are similar to `ggplot2::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 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)

Stats and geoms in this family include:

- `geom_dots()`: dotplots on raw data. Ensures the dotplot fits within available space by reducing the size of the dots automatically (may result in very small dots).
- `geom_swarm()` and `geom_weave()`: dotplots on raw data with defaults intended to create "beeswarm" plots. Used `side = "both"` by default, and sets the default dot size to the same size as `geom_point()` (`binwidth = unit(1.5, "mm")`), allowing dots to overlap instead of getting very small.
- `stat_dots()`: dotplots on raw data, **distributional** objects, and `posterior::rvar()`s
- `geom_dotsinterval()`: dotplot + interval plots on raw data with already-calculated intervals (rarely useful directly).
- `stat_dotsinterval()`: dotplot + interval plots on raw data, **distributional** objects, and `posterior::rvar()`s (will calculate intervals for you).
- `geom_blur_dots()`: blurry dotplots that allow the standard deviation of a blur applied to each dot to be specified using the `sd` aesthetic.
- `stat_mcse_dots()`: blurry dotplots of quantiles using the Monte Carlo Standard Error of each quantile.

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

Value

A ggplot2::Geom representing a dot-weave geometry which can be added to a `ggplot()` object.

Aesthetics

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

Positional aesthetics

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

Dots-specific (aka Slab-specific) aesthetics

- `family`: The font family used to draw the dots.

- order: The order in which data points are stacked within bins. Can be used to create the effect of "stacked" dots by ordering dots according to a discrete variable. If omitted (NULL), the value of the data points themselves are used to determine stacking order. Only applies when layout is "bin" or "hex", as the other layout methods fully determine both *x* and *y* positions.
- side: Which side to place the slab on. "topright", "top", and "right" are synonyms which cause the slab to be drawn on the top or the right depending on if orientation is "horizontal" or "vertical". "bottomleft", "bottom", and "left" are synonyms which cause the slab to be drawn on the bottom or the left depending on if orientation is "horizontal" or "vertical". "topleft" causes the slab to be drawn on the top or the left, and "bottomright" causes the slab to be drawn on the bottom or the right. "both" draws the slab mirrored on both sides (as in a violin plot).
- scale: What proportion of the region allocated to this geom to use to draw the slab. If scale = 1, slabs that use the maximum range will just touch each other. Default is 0.9 to leave some space between adjacent slabs. For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.
- justification: Justification of the interval relative to the slab, where 0 indicates bottom/left justification and 1 indicates top/right justification (depending on orientation). If justification is NULL (the default), then it is set automatically based on the value of side: when side is "top"/"right" justification is set to 0, when side is "bottom"/"left" justification is set to 1, and when side is "both" justification is set to 0.5.
- datatype: When using composite geoms directly without a stat (e.g. geom_slabinterval()), datatype is used to indicate which part of the geom a row in the data targets: rows with datatype = "slab" target the slab portion of the geometry and rows with datatype = "interval" target the interval portion of the geometry. This is set automatically when using ggdist stats.

Interval-specific aesthetics

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

Point-specific aesthetics

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

Color aesthetics

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

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

Line aesthetics

- `linewidth`: Width of the line used to draw the **interval** (except with `geom_slab()`: then it is the width of the **slab**). With composite geometries including an interval and slab, use `slab_linewidth` to set the line width of the **slab** (see below). For **interval**, raw `linewidth` values are transformed according to the `interval_size_domain` and `interval_size_range` parameters of the `geom` (see above).
- `size`: Determines the size of the **point**. If `linewidth` is not provided, `size` will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only `size` and not `linewidth`). Raw `size` values are transformed according to the `interval_size_domain`, `interval_size_range`, and `fatten_point` parameters of the `geom` (see above). Use the `point_size` aesthetic (below) to set sub-geometry size directly without applying the effects of `interval_size_domain`, `interval_size_range`, and `fatten_point`.
- `stroke`: Width of the outline around the **point** sub-geometry.
- `linetype`: Type of line (e.g., `"solid"`, `"dashed"`, etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the `slab_linetype` or `interval_linetype` aesthetics (below) to set sub-geometry line types separately.

Slab-specific color and line override aesthetics

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

Interval-specific color and line override aesthetics

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

Point-specific color and line override aesthetics

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

Deprecated aesthetics

- slab_size: Use slab_linewidth.
- interval_size: Use interval_linewidth.

Other aesthetics (these work as in standard geoms)

- width
- height
- group

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

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 geom_dotsinterval() for the geometry this shortcut is based on.

See vignette("dotsinterval") for a variety of examples of use.

Other dotsinterval geoms: geom_blur_dots(), geom_dots(), geom_dotsinterval(), geom_swarm()

Examples

```
library(dplyr)
library(ggplot2)

theme_set(theme_ggdist())

set.seed(12345)
df = tibble(
  g = rep(c("a", "b"), 200),
  value = rnorm(400, c(0, 3), c(0.75, 1))
)

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

df %>%
  ggplot(aes(x = value, y = g)) +
  geom_weave()

df %>%
  ggplot(aes(y = value, x = g)) +
  geom_weave()
```

ggdist-deprecated *Deprecated functions and arguments in ggdist*

Description

Deprecated functions and arguments and their alternatives are listed below.

Deprecated stats and geoms

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

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

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

Deprecated arguments

Deprecated parameters for `stat_slabinterval()` and family:

- The `.prob` argument, which is a long-deprecated alias for `.width`, was removed in ggdist 3.1.
- The `limits_function` argument: this was a parameter for determining the function to compute limits of the slab in `stat_slabinterval()` and its derived stats. This function is really an internal function only needed by subclasses of the base class, yet added a lot of noise to the documentation, so it was replaced with `AbstractStatSlabInterval$compute_limits()`.
- The `limits_args` argument: extra stat parameters are now passed through to the `...` arguments to `AbstractStatSlabInterval$compute_limits()`; use these instead.
- The `slab_function` argument: this was a parameter for determining the function to compute slabs in `stat_slabinterval()` and its derived stats. This function is really an internal function only needed by subclasses of the base class, yet added a lot of noise to the documentation, so it was replaced with `AbstractStatSlabInterval$compute_slab()`.
- The `slab_args` argument: extra stat parameters are now passed through to the `...` arguments to `AbstractStatSlabInterval$compute_slab()`; use these instead.
- The `slab_type` argument: instead of setting the slab type, either adjust the `density` argument (e.g. use `density = "histogram"` to replace `slab_type = "histogram"`) or use the `pdf` or `cdf` computed variables mapped onto an appropriate aesthetic (e.g. use `aes(thickness = after_stat(cdf))` to create a CDF).
- The `interval_function` and `fun.data` arguments: these were parameters for determining the function to compute intervals in `stat_slabinterval()` and its derived stats. This function is really an internal function only needed by subclasses of the base class, yet added a lot of noise to the documentation, so it was replaced with `AbstractStatSlabInterval$compute_interval()`.
- The `interval_args` and `fun.args` arguments: to pass extra arguments to a `point_interval` replace the value of the `point_interval` argument with a simple wrapper; e.g. `stat_halfeye(point_interval = \(.`

Deprecated parameters for `geom_slabinterval()` and family:

- The `size_domain` and `size_range` arguments, which are long-deprecated aliases for `interval_size_domain` and `interval_size_range`, were removed in ggdist 3.1.

Author(s)

Matthew Kay

`guide_rampbar` *Continuous guide for colour ramp scales (ggplot2 guide)*

Description

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

Usage

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

Arguments

`...` Arguments passed on to `ggplot2::guide_colourbar`

`title` A character string or expression indicating a title of guide. If `NULL`, the title is not shown. By default (`waiver()`), the name of the scale object or the name specified in `labs()` is used for the title.

`theme` A `theme` object to style the guide individually or differently from the plot's theme settings. The `theme` argument in the guide overrides, and is combined with, the plot's theme.

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

`display` A string indicating a method to display the colourbar. Can be one of the following:

- `"raster"` to display as a bitmap image.
- `"rectangles"` to display as a series of rectangles.
- `"gradient"` to display as a linear gradient.

Note that not all devices are able to render rasters and gradients.

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

- `alpha` A numeric between 0 and 1 setting the colour transparency of the bar. Use `NA` to preserve the alpha encoded in the colour itself (default).
- `draw.ulim` A logical specifying if the upper limit tick marks should be visible.
- `draw.llim` A logical specifying if the lower limit tick marks should be visible.
- `position` A character string indicating where the legend should be placed relative to the plot panels.
- `direction` A character string indicating the direction of the guide. One of "horizontal" or "vertical."
- `reverse` logical. If `TRUE` the colourbar is reversed. By default, the highest value is on the top and the lowest value is on the bottom
- `order` positive integer less than 99 that specifies the order of this guide among multiple guides. This controls the order in which multiple guides are displayed, not the contents of the guide itself. If 0 (default), the order is determined by a secret algorithm.

`to` <string> The color to ramp to in the guide. Corresponds to `1` on the scale.

`available_aes` <character> Vector listing the aesthetics for which a `guide_rampbar()` can be drawn.

Details

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

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

Value

A guide object.

Author(s)

Matthew Kay

See Also

Other colour ramp functions: `partial_colour_ramp()`, `ramp_colours()`, `scale_colour_ramp`

Examples

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

# The default guide for ramp scales is guide_legend(), which creates a
# discrete style scale:
```

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

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

# We can tell the guide what color to ramp to using the `to` argument:
tibble(d = dist_uniform(0, 1)) %>%
  ggplot(aes(y = 0, xdist = d)) +
  stat_slab(aes(fill_ramp = after_stat(x)), fill = "blue") +
  scale_fill_ramp_continuous(from = "red", guide = guide_rampbar(to = "blue"))
```

`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 according to an LKJ distribution.

Usage

```
dlkjcorr_marginal(x, K, eta, log = FALSE)

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

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

rlkjcorr_marginal(n, K, eta)
```

Arguments

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

Details

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

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

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

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

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_slabinterval()` family of stats for visualizing them.

Examples

```
library(dplyr)
library(ggplot2)

theme_set(theme_ggdist())

expand.grid(
  eta = 1:6,
  K = 2:6
) %>%
```

```
ggplot(aes(y = ordered(eta), dist = "lkjcorr_marginal", arg1 = K, arg2 = eta)) +
stat_slab() +
facet_grid(~ paste0(K, "x", K)) +
scale_y_discrete(limits = rev) +
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",
  dist_obj = ".dist_obj"
)
```

Arguments

data	<data.frame> 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	<numeric> Dimension of the correlation matrix. Must be greater than or equal to 2.
predicate	<bare language | NULL> 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 that evaluates to `TRUE` on the rows you want to modify. If `NULL` (the default), all `lkjcorr` distributions in `data` are modified.

`dist`	<string> The name of the column containing distribution names. See `parse_dist()`.
`args`	<string> The name of the column containing distribution arguments. See `parse_dist()`.
`dist_obj`	<string> The name of the output column to contain a **distributional** object representing the distribution. 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 (stored in `.args`) is equal to the specified dimension of the correlation matrix (`K`), changes the distribution name in `.dist` to `"lkjcorr_marginal"`, and assigns a **distributional** object representing this distribution to `.dist_obj`. This allows the distribution to be easily visualized using the `stat_slabinterval()` family of ggplot2 stats.

Value

A data frame of the same size and column names as the input, with the `dist`, and `args`, and `dist_obj` 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, xdist = .dist_obj)) +
  stat_halfeye() +
  xlim(-1, 1) +
  xlab("Marginal correlation for LKJ(3) prior on 2x2 correlation matrix")

# Say our prior list has multiple LKJ priors on correlation matrices
# of different sizes, we can supply a predicate expression to select
# only those rows we want to modify
data.frame(coef = c("a", "b"), prior = "lkjcorr(3)") %>%
  parse_dist(prior) %>%
  marginalize_lkjcorr(K = 2, coef == "a") %>%
```

```
marginalize_lkjcorr(K = 4, coef == "b")
```

parse_dist *Parse distribution specifications into columns of a data frame*

Description

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

Usage

```
parse_dist(
  object,
  ...,
  dist = ".dist",
  args = ".args",
  dist_obj = ".dist_obj",
  package = NULL,
  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",
  dist_obj = ".dist_obj",
  package = NULL,
  lb = "lb",
  ub = "ub",
  to_r_names = TRUE
)

## S3 method for class 'character'
parse_dist(
  object,
  ...,
  dist = ".dist",
```

```
  args = ".args",
  dist_obj = ".dist_obj",
  package = NULL,
  to_r_names = TRUE
)

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

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

r_dist_name(dist_name)
```

Arguments

`object` <character | data.frame> One of:

- A character vector containing distribution specifications, like `c("normal(0,1)", "exp(1)")`
- A data frame with a column containing distribution specifications.

`...` Arguments passed to other implementations of `parse_dist()`.

`dist` <string> The name of the output column to contain the distribution name.

`args` <string> The name of the output column to contain the arguments to the distribution.

`dist_obj` <string> The name of the output column to contain a **distributional** object representing the distribution.

`package` <string | environment | NULL> The package or environment to search for distribution functions in. Passed to `distributional::dist_wrap()`. One of:

- a string: use the environment for the package with the given name

- an environment: use the given environment
- `NULL` (default): use the calling environment

`to_r_names` <scalar logical> 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` <bare language> Column or column expression of `object` that resolves to a character vector of distribution specifications (when `object` is a `data.frame()`).

`lb` <string> The name of an input column (for `data.frame` and `brms::prior` objects) that contains the lower bound of the distribution, which if present will produce a truncated distribution using `dist_truncated()`. Ignored if `object[[lb]]` is `NULL` or if it is `NA` for the corresponding input row.

`ub` <string> The name of an input column (for `data.frame` and `brms::prior` objects) that contains the upper bound of the distribution, which if present will produce a truncated distribution using `dist_truncated()`. Ignored if `object[[ub]]` is `NULL` or if it is `NA` for the corresponding input row.

`dist_name` <character> For `r_dist_name()`, a character vector of distribution names to be translated into distribution names R recognizes. Unrecognized names are left as-is.

Details

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

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

Value

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

See Also

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

Examples

```
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 follows the same format as above
```

---

partial_colour_ramp *Partial colour ramp (datatype)*

Description

A representation of a partial ramp between two colours: the origin colour (`from`) and the distance from the origin colour to the target colour (`amount`, a value between `0` and `1`). The target colour of the ramp can be filled in later using `ramp_colours()`, producing a colour.

Usage

```
partial_colour_ramp(amount = double(), from = "white")
```

Arguments

amount	<numeric> Vector of values between `0` and `1` giving amounts to ramp the colour. `0` corresponds to the colour `from`.
from	<character> Vector giving colours to ramp from.

Details

This datatype is used by scale_colour_ramp to create ramped colours in **ggdist** geoms. It is a vctrs::rcrd datatype with two fields: `"amount"`, the amount to ramp, and `"from"`, the colour to ramp from.

Colour ramps can be applied (i.e. translated into colours) using `ramp_colours()`, which can be used with `partial_colour_ramp()` to implement geoms that make use of `colour_ramp` or `fill_ramp` scales.

Value

A vctrs::rcrd of class `"ggdist_partial_colour_ramp"` with fields `"amount"` and `"from"`.

Author(s)

Matthew Kay

See Also

Other colour ramp functions: guide_rampbar(), ramp_colours(), scale_colour_ramp

Examples

```
pcr = partial_colour_ramp(c(0, 0.25, 0.75, 1), "red")
pcr

ramp_colours("blue", pcr)
```

---

point_interval — *Point and interval summaries for tidy data frames of draws from distributions*

Description

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

Supports automatic partial function application.

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 'tbl_df'
point_interval(.data, ...)

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

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

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

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

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

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

```
hdi(
  x,
  .width = 0.95,
  na.rm = FALSE,
  ...,
  density = density_bounded(trim = TRUE),
  n = 4096,
  .prob
)

Mode(x, na.rm = FALSE, ...)

## Default S3 method:
Mode(
  x,
  na.rm = FALSE,
  ...,
  density = density_bounded(trim = TRUE),
  n = 2001,
  weights = NULL
)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Arguments

`.data`	<data.frame | grouped_df> Data frame (or grouped data frame as returned by `dplyr::group_by()`) that contains draws to summarize.
`...`	<bare language> Column names or expressions that, when evaluated in the context of `.data`, represent draws to summarize. If this is empty, then by default all columns that are not group columns and which are not in `.exclude` (by default `".chain"`, `".iteration"`, `".draw"`, and `".row"`) will be summarized. These columns can be numeric, **distributional** objects, `posterior::rvars`, or list columns of numeric values to summarise.
`.width`	<numeric> 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`	<function> Point summary function, which takes a vector and returns a single value, e.g. `mean`, `median`, or `Mode`.
`.interval`	<function> 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`	<scalar logical> 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`	<scalar logical> Should `NA` values 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`	<character> Vector of names of columns to be excluded from summarization if no column names are specified to be summarized in `...`. Default ignores several meta-data column names used in **ggdist** and **tidybayes**.
`.prob`	Deprecated. Use `.width` instead.
`x`	<numeric> Vector to summarize (for interval functions: `qi()`, `hdi()`, etc)

density — <function | string> For hdi() and Mode(), the kernel density estimator to use, either as a function (e.g. density_bounded, density_unbounded) or as a string giving the suffix to a function that starts with density_ (e.g. "bounded" or "unbounded"). The default, "bounded", uses the bounded density estimator of density_bounded(), which itself estimates the bounds of the distribution, and tends to work well on both bounded and unbounded data.

n — <scalar numeric> For hdi() and Mode(), the number of points to use to estimate highest-density intervals or modes.

weights — <numeric | NULL> For Mode(), an optional vector, which (if not NULL) is of the same length as x and provides weights for each element of x.

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.

hdci yields the highest-density *continuous* interval, also known as the shortest probability interval. **Note:** If the distribution is multimodal, this may not actually be the highest-density interval (there may be a higher-density discontinuous interval, which can be found using hdi).

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

Value

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

Author(s)

Matthew Kay

Examples

```
library(dplyr)
library(ggplot2)

set.seed(123)

rnorm(1000) %>%
  median_qi()

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

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

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

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

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

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

position_dodgejust — *Dodge overlapping objects side-to-side, preserving justification*

Description

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

Usage

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

Arguments

`width`	Dodging width, when different to the width of the individual elements. This is useful when you want to align narrow geoms with wider geoms. See the examples.
`preserve`	Should dodging preserve the `"total"` width of all elements at a position, or the width of a `"single"` element?
`justification`	<scalar numeric> Justification of the point position (`x`/`y`) relative to its bounds (`xmin`/`xmax` or `ymin`/`ymax`), where `0` indicates bottom/left justification and `1` indicates top/right justification (depending on `orientation`). This is only used if `xmin`/`xmax`/`ymin`/`ymax` are not supplied; in that case, `justification` will be used along with `width` to determine the bounds of the object prior to dodging.

Examples

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

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

```
# An example with normal "dodge" positioning
# Notice how dodge points are placed in the center of their bounding boxes,
# which can cause slabs to be positioned outside their bounds.
dist_df %>%
  ggplot(aes(
    x = factor(group), ydist = dist_normal(mean, sd),
    fill = subgroup
  )) +
  stat_halfeye(
    position = "dodge"
  ) +
  geom_rect(
    aes(xmin = group, xmax = group + 1, ymin = 2, ymax = 13, color = subgroup),
    position = "dodge",
    data = . %>% filter(group == 3),
    alpha = 0.1
  ) +
  geom_point(
    aes(x = group, y = 7.5, color = subgroup),
    position = position_dodge(width = 1),
    data = . %>% filter(group == 3),
    shape = 1,
    size = 4,
    stroke = 1.5
  ) +
  scale_fill_brewer(palette = "Set2") +
  scale_color_brewer(palette = "Dark2")

# This same example with "dodgejust" positioning. For the points we
# supply a justification parameter to position_dodgejust which mimics the
# justification parameter of stat_halfeye, ensuring that they are
# placed appropriately. On slabinterval family geoms, position_dodgejust()
# will automatically detect the appropriate justification.
dist_df %>%
  ggplot(aes(
    x = factor(group), ydist = dist_normal(mean, sd),
    fill = subgroup
  )) +
  stat_halfeye(
    position = "dodgejust"
  ) +
  geom_rect(
    aes(xmin = group, xmax = group + 1, ymin = 2, ymax = 13, color = subgroup),
    position = "dodgejust",
    data = . %>% filter(group == 3),
    alpha = 0.1
  ) +
  geom_point(
    aes(x = group, y = 7.5, color = subgroup),
    position = position_dodgejust(width = 1, justification = 0),
    data = . %>% filter(group == 3),
    shape = 1,
```

```
    size = 4,
    stroke = 1.5
  ) +
  scale_fill_brewer(palette = "Set2") +
  scale_color_brewer(palette = "Dark2")
```

Pr_ *Probability expressions in ggdist aesthetics*

Description

Experimental probability-like expressions that can be used in place of some `after_stat()` expressions in aesthetic assignments in **ggdist** stats.

Usage

```
Pr_(x)

p_(x)
```

Arguments

x	<bare language> Expressions. See *Probability expressions*, below.

Details

`Pr_()` and `p_()` are an **experimental** mini-language for specifying aesthetic values based on probabilities and probability densities derived from distributions supplied to **ggdist** stats (e.g., in `stat_slabinterval()`, `stat_dotsinterval()`, etc.). They generate expressions that use `after_stat()` and the computed variables of the stat (such as `cdf` and `pdf`; see e.g. the **Computed Variables** section of `stat_slabinterval()`) to compute the desired probabilities or densities.

For example, one way to map the density of a distribution onto the `alpha` aesthetic of a slab is to use `after_stat(pdf)`:

```
ggplot() +
  stat_slab(aes(xdist = distributional::dist_normal(), alpha = after_stat(pdf)))
```

ggdist probability expressions offer an alternative, equivalent syntax:

```
ggplot() +
  stat_slab(aes(xdist = distributional::dist_normal(), alpha = !!p_(x)))
```

Where `p_(x)` is the probability density function. The use of `!!` is necessary to splice the generated expression into the `aes()` call; for more information, see quasiquotation.

Probability expressions

Probability expressions consist of a call to `Pr_()` or `p_()` containing a small number of valid combinations of operators and variable names.

Valid variables in probability expressions include:

- `x`, `y`, or `value`: values along the `x` or `y` axis. `value` is the orientation-neutral form.
- `xdist`, `ydist`, or `dist`: distributions mapped along the `x` or `y` axis. `dist` is the orientation-neutral form. `X` and `Y` can also be used as synonyms for `xdist` and `ydist`.
- `interval`: the smallest interval containing the current `x`/`y` value.

`Pr_()` generates expressions for probabilities, e.g. cumulative distribution functions (CDFs). Valid operators inside `Pr_()` are:

- <, <=, >, >=: generates values of the cumulative distribution function (CDF) or complementary CDF by comparing one of {`x`, `y`, `value`} to one of {`xdist`, `ydist`, `dist`, `X`, `Y`}. For example, `Pr_(xdist <= x)` gives the CDF and `Pr_(xdist > x)` gives the CCDF.
- `%in%`: currently can only be used with `interval` on the right-hand side: gives the probability of {`x`, `y`, `value`} (left-hand side) being in the smallest interval the stat generated that contains the value; e.g. `Pr_(x %in% interval)`.

`p_()` generates expressions for probability density functions or probability mass functions (depending on if the underlying distribution is continuous or discrete). It currently does not allow any operators in the expression, and must be passed one of `x`, `y`, or `value`.

See Also

The *Computed Variables* section of `stat_slabinterval()` (especially `cdf` and `pdf`) and the `after_stat()` function.

Examples

```
library(ggplot2)
library(distributional)

df = data.frame(
  d = c(dist_normal(2.7, 1), dist_lognormal(1, 1/3)),
  name = c("normal", "lognormal")
)

# map density onto alpha of the fill
ggplot(df, aes(y = name, xdist = d)) +
  stat_slabinterval(aes(alpha = !!p_(x)))

# map CCDF onto thickness (like stat_ccdfinterval())
ggplot(df, aes(y = name, xdist = d)) +
  stat_slabinterval(aes(thickness = !!Pr_(xdist > x)))

# map containing interval onto fill
ggplot(df, aes(y = name, xdist = d)) +
  stat_slabinterval(aes(fill = !!Pr_(x %in% interval)))
```

```
# the color scale in the previous example is not great, so turn the
# probability into an ordered factor and adjust the fill scale.
# Though, see also the `level` computed variable in `stat_slabinterval()`,
# which is probably easier to use to create this style of chart.
ggplot(df, aes(y = name, xdist = d)) +
  stat_slabinterval(aes(fill = ordered(!!Pr_(x %in% interval)))) +
  scale_fill_brewer(direction = -1)
```

`ramp_colours` *Apply partial colour ramps*

Description

Given vectors of colours and `partial_colour_ramp`s, ramps the colours according to the parameters of the partial colour ramps, returning a vector of the same length as the inputs giving the transformed (ramped) colours.

Usage

```
ramp_colours(colour, ramp)
```

Arguments

`colour`	<character> Vector of colours to ramp to.
`ramp`	<partial_colour_ramp> Vector of colour ramps (same length as `colour`) giving the colour to ramp from and the amount to ramp.

Details

Takes vectors of colours and `partial_colour_ramp`s and produces colours by interpolating between each `from` colour and the target `colour` the specified `amount` (where `amount` and `from` are the corresponding fields of the `ramp`).

For example, to add support for the `fill_ramp` aesthetic to a geometry, this line could be used inside the `draw_group()` or `draw_panel()` method of a geom:

```
data$fill = ramp_colours(data$fill, data$fill_ramp)
```

Value

A character vector of colours.

Author(s)

Matthew Kay

See Also

Other colour ramp functions: guide_rampbar(), partial_colour_ramp(), scale_colour_ramp

Examples

```
pcr = partial_colour_ramp(c(0, 0.25, 0.75, 1), "red")
pcr

ramp_colours("blue", pcr)
```

---

scale_colour_ramp   *Secondary color scale that ramps from another color (ggplot2 scale)*

Description

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

Usage

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

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

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

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

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

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

Arguments

`from`	<string> The color to ramp from. Corresponds to `0` on the scale.
`...`	Arguments passed to underlying scale or guide functions. E.g. `scale_colour_ramp_discrete()` passes arguments to `discrete_scale()`, `scale_colour_ramp_continuous()` passes arguments to `continuous_scale()`. See those functions for more details.
`limits`	One of: • `NULL` to use the default scale range • A numeric vector of length two providing limits of the scale. Use `NA` to refer to the existing minimum or maximum • A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang lambda function notation. Note that setting limits on positional scales will **remove** data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see `coord_cartesian()`).
`range`	<length-2 numeric> Minimum and maximum values after the scale transformation. These values should be between `0` (the `from` color) and `1` (the color determined by the `fill` aesthetic).
`guide`	<Guide | string> A function used to create a guide or its name. For `scale_colour_ramp_continuous()` and `scale_fill_ramp_continuous()`, `guide_rampbar()` can be used to create gradient color bars. See `guides()` for information on other guides.
`aesthetics`	<character> Names of aesthetics to set scales for.

Details

These scales transform data into `partial_colour_ramp`s. Each `partial_colour_ramp` is a pair of two values: a `from` colour and a numeric `amount` between `0` and `1` representing a distance between `from` and the target color (where `0` indicates the `from` color and `1` the target color).

The target color is determined by the corresponding aesthetic: for example, the `colour_ramp` aesthetic creates ramps between `from` and whatever the value of the `colour` aesthetic is; the `fill_ramp` aesthetic creates ramps between `from` and whatever the value of the `fill` aesthetic is. When the `colour_ramp` aesthetic is set, **ggdist** geometries will modify their `colour` by applying the colour ramp between `from` and `colour` (and similarly for `fill_ramp` and `fill`).

Colour ramps can be applied (i.e. translated into colours) using ramp_colours(), which can be used with partial_colour_ramp() to implement geoms that make use of colour_ramp or fill_ramp scales.

Value

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

Author(s)

Matthew Kay

See Also

Other ggdist scales: scale_side_mirrored(), scale_thickness, sub-geometry-scales

Other colour ramp functions: guide_rampbar(), partial_colour_ramp(), ramp_colours()

Examples

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

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

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

# you can invert the order of `range` to change the order of the blend
tibble(d = dist_normal(0, 1)) %>%
  ggplot(aes(y = 0, xdist = d)) +
  stat_slab(aes(fill_ramp = after_stat(cut_cdf_qi(cdf))), fill = "blue") +
  scale_fill_ramp_discrete(from = "red", range = c(1, 0))
```

---

scale_side_mirrored *Side scale for mirrored slabs (ggplot2 scale)*

Description

This scale creates mirrored slabs for the side aesthetic of the geom_slabinterval() and geom_dotsinterval() family of geoms and stats. It works on discrete variables of two or three levels.

Usage

```
scale_side_mirrored(start = "topright", ..., aesthetics = "side")
```

Arguments

start
: <string> The side to start from. Can be any valid value of the `side` aesthetic except `"both"`.

...
: Arguments passed on to `ggplot2::discrete_scale`

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

  `palette` A palette function that when called with a single integer argument (the number of levels in the scale) returns the values that they should take (e.g., `scales::pal_hue()`).

  `name` The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

  `breaks` One of:
  - `NULL` for no breaks
  - `waiver()` for the default breaks (the scale limits)
  - A character vector of breaks
  - A function that takes the limits as input and returns breaks as output. Also accepts rlang lambda function notation.

  `labels` One of:
  - `NULL` for no labels
  - `waiver()` for the default labels computed by the transformation object
  - A character vector giving labels (must be same length as `breaks`)
  - An expression vector (must be the same length as breaks). See ?plotmath for details.
  - A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

  `limits` One of:
  - `NULL` to use the default scale values
  - A character vector that defines possible values of the scale and their order
  - A function that accepts the existing (automatic) values and returns new ones. Also accepts rlang lambda function notation.

  `expand` For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function `expansion()` to generate the values for the `expand` argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

  `na.translate` Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify `na.translate = FALSE`.

`na.value` If `na.translate = TRUE`, what aesthetic value should the missing values be displayed as? Does not apply to position scales where `NA` is always placed at the far right.

`drop` Should unused factor levels be omitted from the scale? The default, `TRUE`, uses the levels that appear in the data; `FALSE` includes the levels in the factor. Please note that to display every level in a legend, the layer should use `show.legend = TRUE`.

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

`position` For position scales, The position of the axis. `left` or `right` for y axes, `top` or `bottom` for x axes.

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

`super` The super class to use for the constructed scale

`aesthetics` <character> Names of aesthetics to set scales for.

Value

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

Author(s)

Matthew Kay

See Also

Other ggdist scales: `scale_colour_ramp`, `scale_thickness`, `sub-geometry-scales`

Examples

```
library(dplyr)
library(ggplot2)

set.seed(1234)
data.frame(
  x = rnorm(400, c(1,4)),
  g = c("a","b")
) %>%
  ggplot(aes(x, fill = g, side = g)) +
  geom_weave(linewidth = 0, scale = 0.5) +
  scale_side_mirrored()
```

`scale_thickness` *Slab thickness scale (ggplot2 scale)*

Description

This **ggplot2** scale linearly scales all `thickness` values of geoms that support the `thickness` aesthetic (such as `geom_slabinterval()`). It can be used to align the `thickness` scales across multiple geoms (by default, `thickness` is normalized on a per-geom level instead of as a global scale). For a comprehensive discussion and examples of slab scaling and normalization, see the `thickness` scale article.

Usage

```
scale_thickness_shared(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = function(l) c(min(0, l[[1]]), l[[2]]),
  renormalize = FALSE,
  oob = scales::oob_keep,
  guide = "none",
  expand = c(0, 0),
  ...
)

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

Arguments

`name` The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

`breaks` One of:

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

`labels` One of:

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)

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

`limits` One of:

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

`renormalize` <scalar logical> When mapping values to the `thickness` scale, should those values be allowed to be renormalized by geoms (e.g. via the `normalize` parameter to `geom_slabinterval()`)? The default is `FALSE`: if `scale_thickness_shared()` is in use, the geom-specific `normalize` parameter is ignored (this is achieved by flagging values as already normalized by wrapping them in `thickness()`). Set this to `TRUE` to allow geoms to also apply their own normalization. Note that if you set renormalize to `TRUE`, subguides created via the `subguide` parameter to `geom_slabinterval()` will display the scaled values output by this scale, not the original data values.

`oob` One of:

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

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

`expand` <numeric> Vector of limit expansion constants of length 2 or 4, following the same format used by the `expand` argument of `continuous_scale()`. The default is not to expand the limits. You can use the convenience function `expansion()` to generate the expansion values; expanding the lower limit is usually not recommended (because with most `thickness` scales the lower limit is the baseline and represents `0`), so a typical usage might be something like `expand = expansion(c(0, 0.05))` to expand the top end of the scale by 5%.

`...` Arguments passed on to `ggplot2::continuous_scale`

`aesthetics` The names of the aesthetics that this scale works with.

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

`palette` A palette function that when called with a numeric vector with values between 0 and 1 returns the corresponding output values (e.g., `scales::pal_area()`).

`minor_breaks` One of:

- NULL for no minor breaks
- waiver() for the default breaks (one minor break between each major break)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation. When the function has two arguments, it will be given the limits and major breaks.

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

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

na.value Missing values will be replaced with this value.

transform For continuous scales, the name of a transformation object or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "date", "exp", "hms", "identity", "log", "log10", "log1p", "log2", "logit", "modulus", "probability", "probit", "pseudo_log", "reciprocal", "reverse", "sqrt" and "time".

A transformation object bundles together a transform, its inverse, and methods for generating breaks and labels. Transformation objects are defined in the scales package, and are called transform_<name>. If transformations require arguments, you can call them from the scales package, e.g. scales::transform_boxcox(p = 2). You can create your own transformation with scales::new_transform().

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

position For position scales, The position of the axis. left or right for y axes, top or bottom for x axes.

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

super The super class to use for the constructed scale

Details

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

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

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

Value

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

Author(s)

Matthew Kay

See Also

The thickness datatype.

The thickness aesthetic of geom_slabinterval().

subscale_thickness(), for setting a thickness sub-scale within a single geom_slabinterval().

Other ggdist scales: scale_colour_ramp, scale_side_mirrored(), sub-geometry-scales

Examples

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

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

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

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

smooth_density	*Smooth dot positions in a dotplot using a kernel density estimator ("density dotplots")*

Description

Smooths x values using a density estimator, returning new x of the same length. Can be used with a dotplot (e.g. geom_dots(smooth = ...)) to create "density dotplots".

Supports automatic partial function application with waived arguments.

Usage

```
smooth_bounded(
  x,
  density = "bounded",
  bounds = c(NA, NA),
  bounder = "cooke",
  trim = FALSE,
  ...
)

smooth_unbounded(x, density = "unbounded", trim = FALSE, ...)
```

Arguments

x <numeric> Values to smooth.

density <function | string> Density estimator to use for smoothing. One of:

- A function which takes a numeric vector and returns a list with elements x (giving grid points for the density estimator) and y (the corresponding densities). **ggdist** provides a family of functions following this format, including density_unbounded() and density_bounded().
- A string giving the suffix of a function name that starts with "density_"; e.g. "bounded" for [density_bounded()].

bounds <length-2 numeric> Min and max bounds. If a bound is NA, then that bound is estimated from the data using the method specified by bounder.

bounder <function | string> Method to use to find missing (NA) bounds. A function that takes a numeric vector of values and returns a length-2 vector of the estimated lower and upper bound of the distribution. Can also be a string giving the suffix of the name of such a function that starts with "bounder_". Useful values include:

- "cdf": Use the CDF of the the minimum and maximum order statistics of the sample to estimate the bounds. See bounder_cdf().
- "cooke": Use the method from Cooke (1979); i.e. method 2.3 from Loh (1984). See bounder_cooke().
- "range": Use the range of x (i.e the min or max). See bounder_range().

trim — <scalar logical> Passed to density: Should the density estimate be trimmed to the range of the data? Default FALSE.

... — Arguments passed to the density estimator specified by density.

Details

Applies a kernel density estimator (KDE) to x, then uses weighted quantiles of the KDE to generate a new set of x values with smoothed values. Plotted using a dotplot (e.g. geom_dots(smooth = "bounded") or geom_dots(smooth = smooth_bounded(...)), these values create a variation on a "density dotplot" (Zvinca 2018).

Such plots are recommended **only** in very large sample sizes where precise positions of individual values are not particularly meaningful. In small samples, normal dotplots should generally be used.

Two variants are supplied by default:

- smooth_bounded(), which uses density_bounded(). Passes the bounds arguments to the estimator.
- smooth_unbounded(), which uses density_unbounded().

It is generally recommended to pick the smooth based on the known bounds of your data, e.g. by using smooth_bounded() with the bounds parameter if there are finite bounds, or smooth_unbounded() if both bounds are infinite.

Value

A numeric vector of length(x), where each entry is a smoothed version of the corresponding entry in x.

If x is missing, returns a partial application of itself. See automatic-partial-functions.

References

Zvinca, Daniel. "In the pursuit of diversity in data visualization. Jittering data to access details." https://www.linkedin.com/pulse/pursuit-diversity-data-visualization-jittering-access-daniel-zvinca/

See Also

Other dotplot smooths: smooth_discrete(), smooth_none()

Examples

```
library(ggplot2)

set.seed(1234)
x = rnorm(1000)

# basic dotplot is noisy
ggplot(data.frame(x), aes(x)) +
  geom_dots()

# density dotplot is smoother, but does move points (most noticeable
# in areas of low density)
```

```
ggplot(data.frame(x), aes(x)) +
  geom_dots(smooth = "unbounded")

# you can adjust the kernel and bandwidth...
ggplot(data.frame(x), aes(x)) +
  geom_dots(smooth = smooth_unbounded(kernel = "triangular", adjust = 0.5))

# for bounded data, you should use the bounded smoother
x_beta = rbeta(1000, 0.5, 0.5)

ggplot(data.frame(x_beta), aes(x_beta)) +
  geom_dots(smooth = smooth_bounded(bounds = c(0, 1)))
```

`smooth_discrete` *Smooth dot positions in a dotplot of discrete values ("bar dotplots")*

Description

Note: Better-looking bar dotplots are typically easier to achieve using `layout = "bar"` with the `geom_dotsinterval()` family instead of `smooth = "bar"` or `smooth = "discrete"`.

Smooths `x` values where `x` is presumed to be discrete, returning a new `x` of the same length. Both `smooth_discrete()` and `smooth_bar()` use the `resolution()` of the data to apply smoothing around unique values in the dataset; `smooth_discrete()` uses a kernel density estimator and `smooth_bar()` places values in an evenly-spaced grid. Can be used with a dotplot (e.g. `geom_dots(smooth = ...)`) to create "bar dotplots".

Supports automatic partial function application with waived arguments.

Usage

```
smooth_discrete(
  x,
  kernel = c("rectangular", "gaussian", "epanechnikov", "triangular", "biweight",
    "cosine", "optcosine"),
  width = 0.7,
  ...
)

smooth_bar(x, width = 0.7, ...)
```

Arguments

`x`	<numeric> Values to smooth.
`kernel`	<string> The smoothing kernel to be used. This must partially match one of `"gaussian"`, `"rectangular"`, `"triangular"`, `"epanechnikov"`, `"biweight"`, `"cosine"`, or `"optcosine"`. See `stats::density()`.
`width`	<scalar numeric> approximate width of the bars as a fraction of data `resolution()`.

... additional parameters; smooth_discrete() passes these to smooth_unbounded() and thereby to density_unbounded(); smooth_bar() ignores them.

Details

smooth_discrete() applies a kernel density estimator (default: rectangular) to x. It automatically sets the bandwidth to be such that the kernel's width (for each kernel type) is approximately width times the resolution() of the data. This means it essentially creates smoothed bins around each unique value. It calls down to smooth_unbounded().

smooth_bar() generates an evenly-spaced grid of values spanning +/- width/2 around each unique value in x.

Value

A numeric vector of length(x), where each entry is a smoothed version of the corresponding entry in x.

If x is missing, returns a partial application of itself. See automatic-partial-functions.

See Also

Other dotplot smooths: smooth_density, smooth_none()

Examples

```
library(ggplot2)

set.seed(1234)
x = rpois(1000, 2)

# automatic binwidth in basic dotplot on large counts in discrete
# distributions is very small
ggplot(data.frame(x), aes(x)) +
  geom_dots()

# NOTE: It is now recommended to use layout = "bar" instead of
# smooth = "discrete" or smooth = "bar"; the latter are retained because
# they can sometimes be useful in combination with other layouts for
# more specialized (but finicky) applications.
ggplot(data.frame(x), aes(x)) +
  geom_dots(layout = "bar")

# smooth_discrete() constructs wider bins of dots
ggplot(data.frame(x), aes(x)) +
  geom_dots(smooth = "discrete")

# smooth_bar() is an alternative approach to rectangular layouts
ggplot(data.frame(x), aes(x)) +
  geom_dots(smooth = "bar")

# adjust the shape by changing the kernel or the width. epanechnikov
# works well with side = "both"
```

```
ggplot(data.frame(x), aes(x)) +
  geom_dots(smooth = smooth_discrete(kernel = "epanechnikov", width = 0.8), side = "both")
```

`smooth_none` *Apply no smooth to a dotplot*

Description

Default smooth for dotplots: no smooth. Simply returns the input values.

Supports automatic partial function application with waived arguments.

Usage

```
smooth_none(x, ...)
```

Arguments

x	<numeric> Values to smooth.
...	ignored

Details

This is the default value for the `smooth` argument of `geom_dotsinterval()`.

Value

x

If x is missing, returns a partial application of itself. See automatic-partial-functions.

See Also

Other dotplot smooths: `smooth_density`, `smooth_discrete()`

stat_ccdfinterval *CCDF bar plot (shortcut stat)*

Description

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

Roughly equivalent to:

```
stat_slabinterval(
  aes(
    thickness = after_stat(thickness(1 - cdf, 0, 1)),
    justification = after_stat(0.5),
    side = after_stat("topleft")
  ),
  normalize = "none",
  expand = TRUE
)
```

Usage

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

Arguments

`mapping` Set of aesthetic mappings created by `aes()`. 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` <Geom | string> Use to override the default connection between `stat_ccdfinterval()` and `geom_slabinterval()`

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

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

`subscale` <function | string> Sub-scale used to scale values of the `thickness` aesthetic within the groups determined by `normalize`. One of:

- A function that takes an `x` argument giving a numeric vector of values to be scaled and then returns a thickness vector representing the scaled values, such as `subscale_thickness()` or `subscale_identity()`.
- A string giving the name of such a function when prefixed with `"subscale_"`; e.g. `"thickness"` or `"identity"`. The value `"thickness"` using the default subscale, which can be modified by setting `subscale_thickness`; see the documentation for that function.

For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.

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

- `"segments"`: breaks up the slab geometry into segments for each unique combination of fill color and alpha value. This approach is supported by all graphics devices and works well for sharp cutoff values, but can give ugly results if a large number of unique fill colors are being used (as in gradients, like in `stat_gradientinterval()`).
- `"gradient"`: a `grid::linearGradient()` is used to create a smooth gradient fill. This works well for large numbers of unique fill colors, but requires R >= 4.1 and is not yet supported on all graphics devices. As of this writing, the `png()` graphics device with `type = "cairo"`, the

svg() device, the pdf() device, and the ragg::agg_png() devices are known to support this option. On R < 4.1, this option will fall back to fill_type = "segments" with a message.

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

interval_size_domain <length-2 numeric> Minimum and maximum of the values of the size and linewidth aesthetics that will be translated into actual sizes for intervals drawn according to interval_size_range (see the documentation for that argument.)

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

fatten_point <scalar numeric> 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.

arrow <arrow | NULL> Type of arrow heads to use on the interval, or NULL for no arrows.

subguide <function | string> Sub-guide used to annotate the thickness scale. One of:

- A function that takes a scale argument giving a ggplot2::Scale object and an orientation argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as subguide_axis() (to draw a traditional axis) or subguide_none() (to draw no annotation). See subguide_axis() for a list of possibilities and examples.
- A string giving the name of such a function when prefixed with "subguide_"; e.g. "axis" or "none". The values "slab", "dots", and "spike" use the default subguide for their geom families (no subguide), which can

be modified by setting subguide_slab, subguide_dots, or subguide_spike; see the documentation for those functions.

normalize
: <string> Groups within which to scale values of the thickness aesthetic. One of:

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

For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.

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

p_limits
: <length-2 numeric> Probability limits. Used to determine the lower and upper limits of *analytical* distributions (distributions from *samples* ignore this parameter and determine their limits based on the limits of the sample and the value of the trim parameter). E.g., if this is c(.001, .999), then a slab is drawn for the distribution from the quantile at p = .001 to the quantile at p = .999. If the lower (respectively upper) limit is NA, then the lower (upper) limit will be the minimum (maximum) of the distribution's support if it is finite, and 0.001 (0.999) if it is not finite. E.g., if p_limits is c(NA, NA), on a gamma distribution the effective value of p_limits would be c(0, .999) since the gamma distribution is defined on (0, Inf); whereas on a normal distribution it would be equivalent to c(.001, .999) since the normal distribution is defined on (-Inf, Inf).

density
: <function | string> Density estimator for sample data. One of:

- A function which takes a numeric vector and returns a list with elements x (giving grid points for the density estimator) and y (the corresponding densities). **ggdist** provides a family of functions following this format, including density_unbounded() and density_bounded(). This format is also compatible with stats::density().
- A string giving the suffix of a function name that starts with "density_"; e.g. "bounded" for [density_bounded()], "unbounded" for [density_unbounded()], or "histogram" for density_histogram(). Defaults to "bounded", i.e. density_bounded(), which estimates the bounds from the data and then uses a bounded density estimator based on the reflection method.

adjust
: <scalar numeric | waiver> Passed to density (e.g. density_bounded()): Value to multiply the bandwidth of the density estimator by. Default waiver() defers to the default of the density estimator, which is usually 1.

trim
: <scalar logical | waiver> Passed to density (e.g. density_bounded()): Should the density estimate be trimmed to the range of the data? Default waiver() defers to the default of the density estimator, which is usually TRUE.

`breaks` <numeric | function | string | waiver> Passed to density (e.g. density_histogram()): Determines the breakpoints defining bins. Default waiver() defers to the default of the density estimator, which is usually "Scott". Similar to (but not exactly the same as) the breaks argument to graphics::hist(). One of:

- A scalar (length-1) numeric giving the number of bins
- A vector numeric giving the breakpoints between histogram bins
- A function taking x and weights and returning either the number of bins or a vector of breakpoints
- A string giving the suffix of a function that starts with "breaks_". **ggdist** provides weighted implementations of the "Sturges", "Scott", and "FD" break-finding algorithms from graphics::hist(), as well as breaks_fixed() for manually setting the bin width. See breaks.

For example, breaks = "Sturges" will use the breaks_Sturges() algorithm, breaks = 9 will create 9 bins, and breaks = breaks_fixed(width = 1) will set the bin width to 1.

`align` <scalar numeric | function | string | waiver> Passed to density (e.g. density_histogram()): Determines how to align the breakpoints defining bins. Default waiver() defers to the default of the density estimator, which is usually "none" (performs no alignment). One of:

- A scalar (length-1) numeric giving an offset that is subtracted from the breaks. The offset must be between 0 and the bin width.
- A function taking a sorted vector of breaks (bin edges) and returning an offset to subtract from the breaks.
- A string giving the suffix of a function that starts with "align_" used to determine the alignment, such as align_none(), align_boundary(), or align_center().

For example, align = "none" will provide no alignment, align = align_center(at = 0) will center a bin on 0, and align = align_boundary(at = 0) will align a bin edge on 0.

`outline_bars` <scalar logical | waiver> Passed to density (e.g. density_histogram()) and also used for discrete analytical distributions (whose slabs are drawn as histograms). Determines if outlines in between the bars are drawn. If waiver() or FALSE (the default), the outline is drawn only along the tops of the bars. If TRUE, outlines in between bars are also drawn (though you may have to set the slab_color or color aesthetic to see the outlines).

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

`limits` <length-2 numeric> 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` <scalar numeric> Number of points at which to evaluate the function that defines the slab. Also passed to `density` (e.g. `density_bounded()`). Default `waiver()` uses the value `501` for analytical distributions and defers to the default of the density estimator for sample-based distributions, which is also usually `501`.

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

`orientation` <string> Whether this geom is drawn horizontally or vertically. One of:

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

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

`na.rm` <scalar logical> 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? 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). 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

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

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

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

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

Value

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

Computed Variables

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

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

Aesthetics

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

These `stats` support the following aesthetics:

- `x`: x position of the geometry (when orientation = `"vertical"`); or sample data to be summarized (when `orientation = "horizontal"` with sample data).
- `y`: y position of the geometry (when orientation = `"horizontal"`); or sample data to be summarized (when `orientation = "vertical"` with sample data).
- `weight`: When using samples (i.e. the `x` and `y` aesthetics, not `xdist` or `ydist`), optional weights to be applied to each draw.
- `xdist`: When using analytical distributions, distribution to map on the x axis: a **distributional** object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `ydist`: When using analytical distributions, distribution to map on the y axis: a **distributional** object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `dist`: When using analytical distributions, a name of a distribution (e.g. `"norm"`), a **distributional** object (e.g. `dist_normal()`), or a `posterior::rvar()` object. See **Details**.
- `args`: Distribution arguments (`args` or `arg1`, ... `arg9`). See **Details**.

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

Slab-specific aesthetics

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

Interval-specific aesthetics

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

Point-specific aesthetics

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

Color aesthetics

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

Line aesthetics

- linewidth: Width of the line used to draw the **interval** (except with geom_slab(): then it is the width of the **slab**). With composite geometries including an interval and slab, use slab_linewidth to set the line width of the **slab** (see below). For **interval**, raw linewidth values are transformed according to the interval_size_domain and interval_size_range parameters of the geom (see above).
- size: Determines the size of the **point**. If linewidth is not provided, size will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only size and not linewidth). Raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the point_size aesthetic (below) to set sub-geometry size directly without applying the effects of interval_size_domain, interval_size_range, and fatten_point.
- stroke: Width of the outline around the **point** sub-geometry.
- linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the slab_linetype or interval_linetype aesthetics (below) to set sub-geometry line types separately.

Slab-specific color and line override aesthetics

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

- slab_alpha: Override for alpha: the opacity of the slab.
- slab_linewidth: Override for linwidth: the width of the outline of the slab.
- slab_linetype: Override for linetype: the line type of the outline of the slab.

Interval-specific color and line override aesthetics

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

Point-specific color and line override aesthetics

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

Deprecated aesthetics

- slab_size: Use slab_linewidth.
- interval_size: Use interval_linewidth.

Other aesthetics (these work as in standard geoms)

- width
- height
- group

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

See Also

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

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

Examples

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

theme_set(theme_ggdist())

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

# ON ANALYTICAL DISTRIBUTIONS
dist_df = data.frame(
  group = c("a", "b", "c"),
  mean =  c(  5,   7,   8),
  sd =    c(  1, 1.5,   1)
)
# Vectorized distribution types, like distributional::dist_normal()
# and posterior::rvar(), can be used with the `xdist` / `ydist` aesthetics
dist_df %>%
  ggplot(aes(y = group, xdist = dist_normal(mean, sd))) +
  stat_ccdfinterval() +
  expand_limits(x = 0)
```

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

Description

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

Roughly equivalent to:

```
stat_slabinterval(
  aes(
    thickness = after_stat(thickness(cdf, 0, 1)),
    justification = after_stat(0.5),
    side = after_stat("topleft")
  ),
  normalize = "none",
  expand = TRUE
)
```

Usage

```
stat_cdfinterval(
  mapping = NULL,
  data = NULL,
  geom = "slabinterval",
  position = "identity",
  ...,
  normalize = "none",
  expand = TRUE,
  p_limits = c(NA, NA),
  density = "bounded",
  adjust = waiver(),
  trim = waiver(),
  breaks = waiver(),
  align = waiver(),
  outline_bars = waiver(),
  point_interval = "median_qi",
  limits = NULL,
  n = waiver(),
  .width = c(0.66, 0.95),
  orientation = NA,
  na.rm = FALSE,
  show.legend = c(size = FALSE),
  inherit.aes = TRUE
)
```

Arguments

mapping
: Set of aesthetic mappings created by `aes()`. 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
: <Geom | string> Use to override the default connection between `stat_cdfinterval()` and `geom_slabinterval()`

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

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

`subscale` <function | string> Sub-scale used to scale values of the `thickness` aesthetic within the groups determined by `normalize`. One of:

- A function that takes an `x` argument giving a numeric vector of values to be scaled and then returns a thickness vector representing the scaled values, such as `subscale_thickness()` or `subscale_identity()`.
- A string giving the name of such a function when prefixed with `"subscale_"`; e.g. `"thickness"` or `"identity"`. The value `"thickness"` using the default subscale, which can be modified by setting `subscale_thickness`; see the documentation for that function.

For a comprehensive discussion and examples of slab scaling and normalization, see the `thickness` scale article.

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

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

`interval_size_domain` <length-2 numeric> Minimum and maximum of the values of the `size` and `linewidth` aesthetics that will be translated into actual sizes for intervals drawn according to `interval_size_range` (see the documentation for that argument.)

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

fatten_point <scalar numeric> 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.

arrow <arrow | NULL> Type of arrow heads to use on the interval, or NULL for no arrows.

subguide <function | string> Sub-guide used to annotate the thickness scale. One of:

- A function that takes a scale argument giving a ggplot2::Scale object and an orientation argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as subguide_axis() (to draw a traditional axis) or subguide_none() (to draw no annotation). See subguide_axis() for a list of possibilities and examples.
- A string giving the name of such a function when prefixed with "subguide_"; e.g. "axis" or "none". The values "slab", "dots", and "spike" use the default subguide for their geom families (no subguide), which can be modified by setting subguide_slab, subguide_dots, or subguide_spike; see the documentation for those functions.

normalize <string> Groups within which to scale values of the thickness aesthetic. One of:

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

For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.

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

p_limits
: <length-2 numeric> Probability limits. Used to determine the lower and upper limits of *analytical* distributions (distributions from *samples* ignore this parameter and determine their limits based on the limits of the sample and the value of the `trim` parameter). E.g., if this is `c(.001, .999)`, then a slab is drawn for the distribution from the quantile at `p = .001` to the quantile at `p = .999`. If the lower (respectively upper) limit is `NA`, then the lower (upper) limit will be the minimum (maximum) of the distribution's support if it is finite, and `0.001` (`0.999`) if it is not finite. E.g., if `p_limits` is `c(NA, NA)`, on a gamma distribution the effective value of `p_limits` would be `c(0, .999)` since the gamma distribution is defined on `(0, Inf)`; whereas on a normal distribution it would be equivalent to `c(.001, .999)` since the normal distribution is defined on `(-Inf, Inf)`.

density
: <function | string> Density estimator for sample data. One of:
  - A function which takes a numeric vector and returns a list with elements `x` (giving grid points for the density estimator) and `y` (the corresponding densities). **ggdist** provides a family of functions following this format, including `density_unbounded()` and `density_bounded()`. This format is also compatible with `stats::density()`.
  - A string giving the suffix of a function name that starts with `"density_"`; e.g. `"bounded"` for `[density_bounded()]`, `"unbounded"` for `[density_unbounded()]`, or `"histogram"` for `density_histogram()`. Defaults to `"bounded"`, i.e. `density_bounded()`, which estimates the bounds from the data and then uses a bounded density estimator based on the reflection method.

adjust
: <scalar numeric | waiver> Passed to `density` (e.g. `density_bounded()`): Value to multiply the bandwidth of the density estimator by. Default `waiver()` defers to the default of the density estimator, which is usually `1`.

trim
: <scalar logical | waiver> Passed to `density` (e.g. `density_bounded()`): Should the density estimate be trimmed to the range of the data? Default `waiver()` defers to the default of the density estimator, which is usually `TRUE`.

breaks
: <numeric | function | string | waiver> Passed to `density` (e.g. `density_histogram()`): Determines the breakpoints defining bins. Default `waiver()` defers to the default of the density estimator, which is usually `"Scott"`. Similar to (but not exactly the same as) the `breaks` argument to `graphics::hist()`. One of:
  - A scalar (length-1) numeric giving the number of bins
  - A vector numeric giving the breakpoints between histogram bins
  - A function taking `x` and `weights` and returning either the number of bins or a vector of breakpoints
  - A string giving the suffix of a function that starts with `"breaks_"`. **ggdist** provides weighted implementations of the `"Sturges"`, `"Scott"`, and `"FD"` break-finding algorithms from `graphics::hist()`, as well as `breaks_fixed()` for manually setting the bin width. See breaks.

  For example, `breaks = "Sturges"` will use the `breaks_Sturges()` algorithm, `breaks = 9` will create 9 bins, and `breaks = breaks_fixed(width = 1)` will set the bin width to `1`.

align
: <scalar numeric | function | string | waiver> Passed to `density` (e.g. `density_histogram()`): Determines how to align the breakpoints defining bins. Default `waiver()` defers to the default of the density estimator, which is usually `"none"` (performs no alignment). One of:

- A scalar (length-1) numeric giving an offset that is subtracted from the breaks. The offset must be between `0` and the bin width.
- A function taking a sorted vector of `breaks` (bin edges) and returning an offset to subtract from the breaks.
- A string giving the suffix of a function that starts with `"align_"` used to determine the alignment, such as `align_none()`, `align_boundary()`, or `align_center()`.

For example, `align = "none"` will provide no alignment, `align = align_center(at = 0)` will center a bin on `0`, and `align = align_boundary(at = 0)` will align a bin edge on `0`.

`outline_bars` <scalar logical | waiver> Passed to `density` (e.g. `density_histogram()`) and also used for discrete analytical distributions (whose slabs are drawn as histograms). Determines if outlines in between the bars are drawn. If `waiver()` or `FALSE` (the default), the outline is drawn only along the tops of the bars. If `TRUE`, outlines in between bars are also drawn (though you may have to set the `slab_color` or `color` aesthetic to see the outlines).

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

`limits` <length-2 numeric> 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` <scalar numeric> Number of points at which to evaluate the function that defines the slab. Also passed to `density` (e.g. `density_bounded()`). Default `waiver()` uses the value `501` for analytical distributions and defers to the default of the density estimator for sample-based distributions, which is also usually `501`.

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

`orientation` <string> Whether this geom is drawn horizontally or vertically. One of:

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

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

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

na.rm — <scalar logical> 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? 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). 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

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

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

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

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

Value

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

Computed Variables

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

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

Aesthetics

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

These `stats` support the following aesthetics:

- `x`: x position of the geometry (when orientation = `"vertical"`); or sample data to be summarized (when `orientation` = `"horizontal"` with sample data).
- `y`: y position of the geometry (when orientation = `"horizontal"`); or sample data to be summarized (when `orientation` = `"vertical"` with sample data).
- `weight`: When using samples (i.e. the `x` and `y` aesthetics, not `xdist` or `ydist`), optional weights to be applied to each draw.
- `xdist`: When using analytical distributions, distribution to map on the x axis: a **distributional** object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `ydist`: When using analytical distributions, distribution to map on the y axis: a **distributional** object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `dist`: When using analytical distributions, a name of a distribution (e.g. `"norm"`), a **distributional** object (e.g. `dist_normal()`), or a `posterior::rvar()` object. See **Details**.
- `args`: Distribution arguments (`args` or `arg1`, ... `arg9`). See **Details**.

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

Slab-specific aesthetics

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

Interval-specific aesthetics

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

Point-specific aesthetics

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

Color aesthetics

- colour: (or color) The color of the **interval** and **point** sub-geometries. Use the slab_color, interval_color, or point_color aesthetics (below) to set sub-geometry colors separately.
- fill: The fill color of the **slab** and **point** sub-geometries. Use the slab_fill or point_fill aesthetics (below) to set sub-geometry colors separately.
- alpha: The opacity of the **slab**, **interval**, and **point** sub-geometries. Use the slab_alpha, interval_alpha, or point_alpha aesthetics (below) to set sub-geometry colors separately.

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

Line aesthetics

- linewidth: Width of the line used to draw the **interval** (except with geom_slab(): then it is the width of the **slab**). With composite geometries including an interval and slab, use slab_linewidth to set the line width of the **slab** (see below). For **interval**, raw linewidth values are transformed according to the interval_size_domain and interval_size_range parameters of the geom (see above).
- size: Determines the size of the **point**. If linewidth is not provided, size will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only size and not linewidth). Raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the point_size aesthetic (below) to set sub-geometry size directly without applying the effects of interval_size_domain, interval_size_range, and fatten_point.
- stroke: Width of the outline around the **point** sub-geometry.
- linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the slab_linetype or interval_linetype aesthetics (below) to set sub-geometry line types separately.

Slab-specific color and line override aesthetics

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

Interval-specific color and line override aesthetics

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

Point-specific color and line override aesthetics

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

Deprecated aesthetics

- slab_size: Use slab_linewidth.
- interval_size: Use interval_linewidth.

Other aesthetics (these work as in standard geoms)

- width
- height
- group

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

See Also

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

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

Examples

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

theme_set(theme_ggdist())

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

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

stat_dots *Dot plot (shortcut stat)*

Description

A combination of stat_slabinterval() and geom_dotsinterval() with sensible defaults for making dot plots. While geom_dotsinterval() is intended for use on data frames that have already been summarized using a point_interval() function, stat_dots() is intended for use directly on data frames of draws or of analytical distributions, and will perform the summarization using a point_interval() function. Geoms based on geom_dotsinterval() create dotplots that automatically determine a bin width that ensures the plot fits within the available space. They can also ensure dots do not overlap.

Roughly equivalent to:

```
stat_dotsinterval(
  aes(size = NULL),
  geom = "dots",
  show_point = FALSE,
  show_interval = FALSE,
  show.legend = NA
)
```

Usage

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

Arguments

`mapping` Set of aesthetic mappings created by aes(). 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` <Geom | string> Use to override the default connection between `stat_dots()` and `geom_dots()`

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

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

`binwidth` <numeric | unit> The bin width to use for laying out the dots. One of:

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

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

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

`stackratio` <scalar numeric> The distance between the center of the dots in the same stack relative to the dot height. The default, `1`, makes dots in the same stack just touch each other.

`layout` <string> The layout method used for the dots. One of:

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

- `"weave"`: uses the same basic binning approach of `"bin"`, but places dots in the off-axis at their actual positions (unless `overlaps = "nudge"`, in which case overlaps may be nudged out of the way). This maintains the alignment of rows but does not align dots within columns.
- `"hex"`: uses the same basic binning approach of `"bin"`, but alternates placing dots `+ binwidth/4` or `- binwidth/4` in the off-axis from the bin center. This allows hexagonal packing by setting a `stackratio` less than 1 (something like `0.9` tends to work).
- `"swarm"`: uses the `"compactswarm"` layout from `beeswarm::beeswarm()`. Does not maintain alignment of rows or columns, but can be more compact and neat looking, especially for sample data (as opposed to quantile dotplots of theoretical distributions, which may look better with `"bin"`, `"weave"`, or `"hex"`).
- `"bar"`: for discrete distributions, lays out duplicate values in rectangular bars.

`overlaps` <string> How to handle overlapping dots or bins in the `"bin"`, `"weave"`, and `"hex"` layouts (dots never overlap in the `"swarm"` or `"bar"` layouts). For the purposes of this argument, dots are only considered to be overlapping if they would be overlapping when `dotsize = 1` and `stackratio = 1`; i.e. if you set those arguments to other values, overlaps may still occur. One of:

- `"keep"`: leave overlapping dots as they are. Dots may overlap (usually only slightly) in the `"bin"`, `"weave"`, and `"hex"` layouts.
- `"nudge"`: nudge overlapping dots out of the way. Overlaps are avoided using a constrained optimization which minimizes the squared distance of dots to their desired positions, subject to the constraint that adjacent dots do not overlap.

`smooth` <function | string> Smoother to apply to dot positions. One of:

- A function that takes a numeric vector of dot positions and returns a smoothed version of that vector, such as `smooth_bounded()`, `smooth_unbounded()`, smooth_discrete()`, or `smooth_bar()‘.
- A string indicating what smoother to use, as the suffix to a function name starting with `smooth_`; e.g. `"none"` (the default) applies `smooth_none()`, which simply returns the given vector without applying smoothing.

Smoothing is most effective when the smoother is matched to the support of the distribution; e.g. using `smooth_bounded(bounds = ...)`.

`overflow` <string> How to handle overflow of dots beyond the extent of the geom when a minimum `binwidth` (or an exact `binwidth`) is supplied. One of:

- `"keep"`: Keep the overflow, drawing dots outside the geom bounds.
- `"warn"`: Keep the overflow, but produce a warning suggesting solutions, such as setting `binwidth = NA` or `overflow = "compress"`.

- "compress": Compress the layout. Reduces the binwidth to the size necessary to keep the dots within bounds, then adjusts stackratio and dotsize so that the apparent dot size is the user-specified minimum binwidth times the user-specified dotsize.

If you find the default layout has dots that are too small, and you are okay with dots overlapping, consider setting overflow = "compress" and supplying an exact or minimum dot size using binwidth.

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

subguide <function | string> Sub-guide used to annotate the thickness scale. One of:

- A function that takes a scale argument giving a ggplot2::Scale object and an orientation argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as subguide_axis() (to draw a traditional axis) or subguide_none() (to draw no annotation). See subguide_axis() for a list of possibilities and examples.
- A string giving the name of such a function when prefixed with "subguide_"; e.g. "axis" or "none". The values "slab", "dots", and "spike" use the default subguide for their geom families (no subguide), which can be modified by setting subguide_slab, subguide_dots, or subguide_spike; see the documentation for those functions.

quantiles <scalar logical> Number of quantiles to plot in the dotplot. Use NA (the default) to plot all data points. Setting this to a value other than NA will produce a quantile dotplot: that is, a dotplot of quantiles from the sample or distribution (for analytical distributions, the default of NA is taken to mean 100 quantiles). See Kay et al. (2016) and Fernandes et al. (2018) for more information on quantile dotplots.

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

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

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

`na.rm`	<scalar logical> 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

The *dots* family of stats and geoms are similar to `ggplot2::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 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)

Stats and geoms in this family include:

- `geom_dots()`: dotplots on raw data. Ensures the dotplot fits within available space by reducing the size of the dots automatically (may result in very small dots).
- `geom_swarm()` and `geom_weave()`: dotplots on raw data with defaults intended to create "beeswarm" plots. Used `side = "both"` by default, and sets the default dot size to the same size as `geom_point()` (`binwidth = unit(1.5, "mm")`), allowing dots to overlap instead of getting very small.
- `stat_dots()`: dotplots on raw data, **distributional** objects, and `posterior::rvar()`s
- `geom_dotsinterval()`: dotplot + interval plots on raw data with already-calculated intervals (rarely useful directly).
- `stat_dotsinterval()`: dotplot + interval plots on raw data, **distributional** objects, and `posterior::rvar()`s (will calculate intervals for you).
- `geom_blur_dots()`: blurry dotplots that allow the standard deviation of a blur applied to each dot to be specified using the `sd` aesthetic.
- `stat_mcse_dots()`: blurry dotplots of quantiles using the Monte Carlo Standard Error of each quantile.

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

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

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

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

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

Value

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

Computed Variables

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

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

- f: (deprecated) For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by slab_type. Instead of using slab_type to change f and then mapping f onto an aesthetic, it is now recommended to simply map the corresponding computed variable (e.g. pdf, cdf, or 1 - cdf) directly onto the desired aesthetic.

Aesthetics

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

These stats support the following aesthetics:

- x: x position of the geometry (when orientation = "vertical"); or sample data to be summarized (when orientation = "horizontal" with sample data).
- y: y position of the geometry (when orientation = "horizontal"); or sample data to be summarized (when orientation = "vertical" with sample data).
- weight: When using samples (i.e. the x and y aesthetics, not xdist or ydist), optional weights to be applied to each draw.
- xdist: When using analytical distributions, distribution to map on the x axis: a **distributional** object (e.g. dist_normal()) or a posterior::rvar() object.
- ydist: When using analytical distributions, distribution to map on the y axis: a **distributional** object (e.g. dist_normal()) or a posterior::rvar() object.
- dist: When using analytical distributions, a name of a distribution (e.g. "norm"), a **distributional** object (e.g. dist_normal()), or a posterior::rvar() object. See **Details**.
- args: Distribution arguments (args or arg1, ... arg9). See **Details**.

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

Dots-specific (aka Slab-specific) aesthetics

- family: The font family used to draw the dots.
- order: The order in which data points are stacked within bins. Can be used to create the effect of "stacked" dots by ordering dots according to a discrete variable. If omitted (NULL), the value of the data points themselves are used to determine stacking order. Only applies when layout is "bin" or "hex", as the other layout methods fully determine both *x* and *y* positions.
- side: Which side to place the slab on. "topright", "top", and "right" are synonyms which cause the slab to be drawn on the top or the right depending on if orientation is "horizontal" or "vertical". "bottomleft", "bottom", and "left" are synonyms which cause the slab to be drawn on the bottom or the left depending on if orientation is "horizontal" or "vertical". "topleft" causes the slab to be drawn on the top or the left, and "bottomright" causes the slab to be drawn on the bottom or the right. "both" draws the slab mirrored on both sides (as in a violin plot).
- scale: What proportion of the region allocated to this geom to use to draw the slab. If scale = 1, slabs that use the maximum range will just touch each other. Default is 0.9 to leave some space between adjacent slabs. For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.

- justification: Justification of the interval relative to the slab, where 0 indicates bottom/left justification and 1 indicates top/right justification (depending on orientation). If justification is NULL (the default), then it is set automatically based on the value of side: when side is "top"/"right" justification is set to 0, when side is "bottom"/"left" justification is set to 1, and when side is "both" justification is set to 0.5.
- datatype: When using composite geoms directly without a stat (e.g. geom_slabinterval()), datatype is used to indicate which part of the geom a row in the data targets: rows with datatype = "slab" target the slab portion of the geometry and rows with datatype = "interval" target the interval portion of the geometry. This is set automatically when using ggdist stats.

Interval-specific aesthetics

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

Point-specific aesthetics

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

Color aesthetics

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

Line aesthetics

- linewidth: Width of the line used to draw the **interval** (except with geom_slab(): then it is the width of the **slab**). With composite geometries including an interval and slab, use slab_linewidth to set the line width of the **slab** (see below). For **interval**, raw linewidth values are transformed according to the interval_size_domain and interval_size_range parameters of the geom (see above).
- size: Determines the size of the **point**. If linewidth is not provided, size will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only size and not linewidth). Raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the point_size aesthetic (below) to set sub-geometry size directly without applying the effects of interval_size_domain, interval_size_range, and fatten_point.

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

Slab-specific color and line override aesthetics

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

Interval-specific color and line override aesthetics

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

Point-specific color and line override aesthetics

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

Deprecated aesthetics

- slab_size: Use slab_linewidth.
- interval_size: Use interval_linewidth.

Other aesthetics (these work as in standard geoms)

- width
- height
- group

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

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 `geom_dots()` for the geom underlying this stat. See `vignette("dotsinterval")` for a variety of examples of use.

Other dotsinterval stats: `stat_dotsinterval()`, `stat_mcse_dots()`

Examples

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

theme_set(theme_ggdist())

# ON SAMPLE DATA
set.seed(12345)
tibble(
  x = rep(1:10, 100),
  y = rnorm(1000, x)
) %>%
  ggplot(aes(x = x, y = y)) +
  stat_dots()

# ON ANALYTICAL DISTRIBUTIONS
# Vectorized distribution types, like distributional::dist_normal()
# and posterior::rvar(), can be used with the `xdist` / `ydist` aesthetics
tibble(
  x = 1:10,
  sd = seq(1, 3, length.out = 10)
) %>%
  ggplot(aes(x = x, ydist = dist_normal(x, sd))) +
  stat_dots(quantiles = 50)
```

`stat_dotsinterval` *Dots + point + interval plot (shortcut stat)*

Description

A combination of stat_slabinterval() and geom_dotsinterval() with sensible defaults for making dots + point + interval plots. While geom_dotsinterval() is intended for use on data frames that have already been summarized using a point_interval() function, stat_dotsinterval() is intended for use directly on data frames of draws or of analytical distributions, and will perform the summarization using a point_interval() function. Geoms based on geom_dotsinterval() create dotplots that automatically determine a bin width that ensures the plot fits within the available space. They can also ensure dots do not overlap.

Usage

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

Arguments

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

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

`binwidth` <numeric | unit> The bin width to use for laying out the dots. One of:

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

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

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

`stackratio` <scalar numeric> The distance between the center of the dots in the same stack relative to the dot height. The default, `1`, makes dots in the same stack just touch each other.

`layout` <string> The layout method used for the dots. One of:

- `"bin"` (default): places dots on the off-axis at the midpoint of their bins as in the classic Wilkinson dotplot. This maintains the alignment of rows and columns in the dotplot. This layout is slightly different from the classic Wilkinson algorithm in that: (1) it nudges bins slightly to avoid overlapping bins and (2) if the input data are symmetrical it will return a symmetrical layout.
- `"weave"`: uses the same basic binning approach of `"bin"`, but places dots in the off-axis at their actual positions (unless `overlaps = "nudge"`, in which case overlaps may be nudged out of the way). This maintains the alignment of rows but does not align dots within columns.
- `"hex"`: uses the same basic binning approach of `"bin"`, but alternates placing dots `+ binwidth/4` or `- binwidth/4` in the off-axis from the bin center. This allows hexagonal packing by setting a `stackratio` less than 1 (something like `0.9` tends to work).

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

`overlaps` <string> How to handle overlapping dots or bins in the `"bin"`, `"weave"`, and `"hex"` layouts (dots never overlap in the `"swarm"` or `"bar"` layouts). For the purposes of this argument, dots are only considered to be overlapping if they would be overlapping when `dotsize = 1` and `stackratio = 1`; i.e. if you set those arguments to other values, overlaps may still occur. One of:

- `"keep"`: leave overlapping dots as they are. Dots may overlap (usually only slightly) in the `"bin"`, `"weave"`, and `"hex"` layouts.
- `"nudge"`: nudge overlapping dots out of the way. Overlaps are avoided using a constrained optimization which minimizes the squared distance of dots to their desired positions, subject to the constraint that adjacent dots do not overlap.

`smooth` <function | string> Smoother to apply to dot positions. One of:

- A function that takes a numeric vector of dot positions and returns a smoothed version of that vector, such as `smooth_bounded()`, `smooth_unbounded()`, smooth_discrete()`, or `smooth_bar()‘.
- A string indicating what smoother to use, as the suffix to a function name starting with `smooth_`; e.g. `"none"` (the default) applies `smooth_none()`, which simply returns the given vector without applying smoothing.

Smoothing is most effective when the smoother is matched to the support of the distribution; e.g. using `smooth_bounded(bounds = ...)`.

`overflow` <string> How to handle overflow of dots beyond the extent of the geom when a minimum `binwidth` (or an exact `binwidth`) is supplied. One of:

- `"keep"`: Keep the overflow, drawing dots outside the geom bounds.
- `"warn"`: Keep the overflow, but produce a warning suggesting solutions, such as setting `binwidth = NA` or `overflow = "compress"`.
- `"compress"`: Compress the layout. Reduces the `binwidth` to the size necessary to keep the dots within bounds, then adjusts `stackratio` and `dotsize` so that the apparent dot size is the user-specified minimum `binwidth` times the user-specified `dotsize`.

If you find the default layout has dots that are too small, and you are okay with dots overlapping, consider setting `overflow = "compress"` and supplying an exact or minimum dot size using `binwidth`.

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

`interval_size_domain` <length-2 numeric> Minimum and maximum of the values of the `size` and `linewidth` aesthetics that will be translated into actual sizes for intervals drawn according to `interval_size_range` (see the documentation for that argument.)

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

`fatten_point` <scalar numeric> 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`.

`arrow` <arrow | NULL> Type of arrow heads to use on the interval, or `NULL` for no arrows.

`subguide` <function | string> Sub-guide used to annotate the `thickness` scale. One of:

- A function that takes a `scale` argument giving a ggplot2::Scale object and an `orientation` argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as `subguide_axis()` (to draw a traditional axis) or `subguide_none()` (to draw no annotation). See `subguide_axis()` for a list of possibilities and examples.
- A string giving the name of such a function when prefixed with `"subguide_"`; e.g. `"axis"` or `"none"`. The values `"slab"`, `"dots"`, and `"spike"` use the default subguide for their geom families (no subguide), which can be modified by setting `subguide_slab`, `subguide_dots`, or `subguide_spike`; see the documentation for those functions.

`quantiles` <scalar logical> Number of quantiles to plot in the dotplot. Use `NA` (the default) to plot all data points. Setting this to a value other than `NA` will produce a quantile dotplot: that is, a dotplot of quantiles from the sample or distribution (for analytical distributions, the default of `NA` is taken to mean `100` quantiles). See Kay et al. (2016) and Fernandes et al. (2018) for more information on quantile dotplots.

`point_interval` <function | string> A function from the `point_interval()` family (e.g., `median_qi`,

	`mean_qi`, `mode_hdi`, etc), or a string giving the name of a function from that family (e.g., `"median_qi"`, `"mean_qi"`, `"mode_hdi"`, etc; if a string, the caller's environment is searched for the function, followed by the **ggdist** environment). This function determines the point summary (typically mean, median, or mode) and interval type (quantile interval, `qi`; highest-density interval, `hdi`; or highest-density continuous interval, `hdci`). Output will be converted to the appropriate `x`- or `y`-based aesthetics depending on the value of `orientation`. See the `point_interval()` family of functions for more information.
`.width`	<numeric> The `.width` argument passed to `point_interval`: a vector of probabilities to use that determine the widths of the resulting intervals. If multiple probabilities are provided, multiple intervals per group are generated, each with a different probability interval (and value of the corresponding `.width` and `level` generated variables).
`orientation`	<string> Whether this geom is drawn horizontally or vertically. One of: • `NA` (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time. • `"horizontal"` (or `"y"`): draw horizontally, using the `y` aesthetic to identify different groups. For each group, uses the `x`, `xmin`, `xmax`, and `thickness` aesthetics to draw points, intervals, and slabs. • `"vertical"` (or `"x"`): draw vertically, using the `x` aesthetic to identify different groups. For each group, uses the `y`, `ymin`, `ymax`, and `thickness` aesthetics to draw points, intervals, and slabs. For compatibility with the base ggplot naming scheme for `orientation`, `"x"` can be used as an alias for `"vertical"` and `"y"` as an alias for `"horizontal"` (**ggdist** had an `orientation` parameter before base ggplot did, hence the discrepancy).
`na.rm`	<scalar logical> 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

The *dots* family of stats and geoms are similar to `ggplot2::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 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)

Stats and geoms in this family include:

- geom_dots(): dotplots on raw data. Ensures the dotplot fits within available space by reducing the size of the dots automatically (may result in very small dots).
- geom_swarm() and geom_weave(): dotplots on raw data with defaults intended to create "beeswarm" plots. Used side = "both" by default, and sets the default dot size to the same size as geom_point() (binwidth = unit(1.5, "mm")), allowing dots to overlap instead of getting very small.
- stat_dots(): dotplots on raw data, **distributional** objects, and posterior::rvar()s
- geom_dotsinterval(): dotplot + interval plots on raw data with already-calculated intervals (rarely useful directly).
- stat_dotsinterval(): dotplot + interval plots on raw data, **distributional** objects, and posterior::rvar()s (will calculate intervals for you).
- geom_blur_dots(): blurry dotplots that allow the standard deviation of a blur applied to each dot to be specified using the sd aesthetic.
- stat_mcse_dots(): blurry dotplots of quantiles using the Monte Carlo Standard Error of each quantile.

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

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

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

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

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

Value

A ggplot2::Stat representing a dots + point + interval geometry which can be added to a `ggplot()` object.

Computed Variables

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

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

Aesthetics

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

These `stats` support the following aesthetics:

- `x`: x position of the geometry (when orientation = `"vertical"`); or sample data to be summarized (when `orientation` = `"horizontal"` with sample data).
- `y`: y position of the geometry (when orientation = `"horizontal"`); or sample data to be summarized (when `orientation` = `"vertical"` with sample data).
- `weight`: When using samples (i.e. the `x` and `y` aesthetics, not `xdist` or `ydist`), optional weights to be applied to each draw.
- `xdist`: When using analytical distributions, distribution to map on the x axis: a **distributional** object (e.g. `dist_normal()`) or a `posterior::rvar()` object.

- ydist: When using analytical distributions, distribution to map on the y axis: a **distributional** object (e.g. dist_normal()) or a posterior::rvar() object.
- dist: When using analytical distributions, a name of a distribution (e.g. "norm"), a **distributional** object (e.g. dist_normal()), or a posterior::rvar() object. See **Details**.
- args: Distribution arguments (args or arg1, ... arg9). See **Details**.

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

Dots-specific (aka Slab-specific) aesthetics

- family: The font family used to draw the dots.
- order: The order in which data points are stacked within bins. Can be used to create the effect of "stacked" dots by ordering dots according to a discrete variable. If omitted (NULL), the value of the data points themselves are used to determine stacking order. Only applies when layout is "bin" or "hex", as the other layout methods fully determine both *x* and *y* positions.
- side: Which side to place the slab on. "topright", "top", and "right" are synonyms which cause the slab to be drawn on the top or the right depending on if orientation is "horizontal" or "vertical". "bottomleft", "bottom", and "left" are synonyms which cause the slab to be drawn on the bottom or the left depending on if orientation is "horizontal" or "vertical". "topleft" causes the slab to be drawn on the top or the left, and "bottomright" causes the slab to be drawn on the bottom or the right. "both" draws the slab mirrored on both sides (as in a violin plot).
- scale: What proportion of the region allocated to this geom to use to draw the slab. If scale = 1, slabs that use the maximum range will just touch each other. Default is 0.9 to leave some space between adjacent slabs. For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.
- justification: Justification of the interval relative to the slab, where 0 indicates bottom/left justification and 1 indicates top/right justification (depending on orientation). If justification is NULL (the default), then it is set automatically based on the value of side: when side is "top"/"right" justification is set to 0, when side is "bottom"/"left" justification is set to 1, and when side is "both" justification is set to 0.5.
- datatype: When using composite geoms directly without a stat (e.g. geom_slabinterval()), datatype is used to indicate which part of the geom a row in the data targets: rows with datatype = "slab" target the slab portion of the geometry and rows with datatype = "interval" target the interval portion of the geometry. This is set automatically when using ggdist stats.

Interval-specific aesthetics

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

Point-specific aesthetics

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

Color aesthetics

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

Line aesthetics

- linewidth: Width of the line used to draw the **interval** (except with geom_slab(): then it is the width of the **slab**). With composite geometries including an interval and slab, use slab_linewidth to set the line width of the **slab** (see below). For **interval**, raw linewidth values are transformed according to the interval_size_domain and interval_size_range parameters of the geom (see above).
- size: Determines the size of the **point**. If linewidth is not provided, size will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only size and not linewidth). Raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the point_size aesthetic (below) to set sub-geometry size directly without applying the effects of interval_size_domain, interval_size_range, and fatten_point.
- stroke: Width of the outline around the **point** sub-geometry.
- linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the slab_linetype or interval_linetype aesthetics (below) to set sub-geometry line types separately.

Slab-specific color and line override aesthetics

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

Interval-specific color and line override aesthetics

- interval_colour: (or interval_color) Override for colour/color: the color of the interval.
- interval_alpha: Override for alpha: the opacity of the interval.

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

Point-specific color and line override aesthetics

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

Deprecated aesthetics

- `slab_size`: Use `slab_linewidth`.
- `interval_size`: Use `interval_linewidth`.

Other aesthetics (these work as in standard geoms)

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

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

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 `geom_dotsinterval()` for the geom underlying this stat. See `vignette("dotsinterval")` for a variety of examples of use.

Other dotsinterval stats: `stat_dots()`, `stat_mcse_dots()`

Examples

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

theme_set(theme_ggdist())

# ON SAMPLE DATA
set.seed(12345)
```

```
tibble(
  x = rep(1:10, 100),
  y = rnorm(1000, x)
) %>%
  ggplot(aes(x = x, y = y)) +
  stat_dotsinterval()

# ON ANALYTICAL DISTRIBUTIONS
# Vectorized distribution types, like distributional::dist_normal()
# and posterior::rvar(), can be used with the `xdist` / `ydist` aesthetics
tibble(
  x = 1:10,
  sd = seq(1, 3, length.out = 10)
) %>%
  ggplot(aes(x = x, ydist = dist_normal(x, sd))) +
  stat_dotsinterval(quantiles = 50)
```

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

Description

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

Roughly equivalent to:

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

Usage

```
stat_eye(
  mapping = NULL,
  data = NULL,
  geom = "slabinterval",
  position = "identity",
  ...,
  p_limits = c(NA, NA),
  density = "bounded",
  adjust = waiver(),
  trim = waiver(),
  breaks = waiver(),
  align = waiver(),
  outline_bars = waiver(),
  expand = FALSE,
  point_interval = "median_qi",
  limits = NULL,
```

```
    n = waiver(),
    .width = c(0.66, 0.95),
    orientation = NA,
    na.rm = FALSE,
    show.legend = c(size = FALSE),
    inherit.aes = TRUE
  )
```

Arguments

`mapping` Set of aesthetic mappings created by `aes()`. 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` <Geom | string> Use to override the default connection between `stat_eye()` and `geom_slabinterval()`

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

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

`subscale` <function | string> Sub-scale used to scale values of the `thickness` aesthetic within the groups determined by `normalize`. One of:

- A function that takes an `x` argument giving a numeric vector of values to be scaled and then returns a thickness vector representing the scaled values, such as `subscale_thickness()` or `subscale_identity()`.
- A string giving the name of such a function when prefixed with `"subscale_"`; e.g. `"thickness"` or `"identity"`. The value `"thickness"` using the default subscale, which can be modified by setting `subscale_thickness`; see the documentation for that function.

For a comprehensive discussion and examples of slab scaling and normalization, see the `thickness` scale article.

`normalize` <string> Groups within which to scale values of the `thickness` aesthetic. One of:

- `"all"`: normalize so that the maximum height across all data is `1`.

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

For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.

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

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

interval_size_domain <length-2 numeric> Minimum and maximum of the values of the size and linewidth aesthetics that will be translated into actual sizes for intervals drawn according to interval_size_range (see the documentation for that argument.)

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

`fatten_point` <scalar numeric> 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`.

`arrow` <arrow | NULL> Type of arrow heads to use on the interval, or `NULL` for no arrows.

`subguide` <function | string> Sub-guide used to annotate the `thickness` scale. One of:

- A function that takes a `scale` argument giving a ggplot2::Scale object and an `orientation` argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as `subguide_axis()` (to draw a traditional axis) or `subguide_none()` (to draw no annotation). See `subguide_axis()` for a list of possibilities and examples.
- A string giving the name of such a function when prefixed with `"subguide_"`; e.g. `"axis"` or `"none"`. The values `"slab"`, `"dots"`, and `"spike"` use the default subguide for their geom families (no subguide), which can be modified by setting `subguide_slab`, `subguide_dots`, or `subguide_spike`; see the documentation for those functions.

`p_limits` <length-2 numeric> Probability limits. Used to determine the lower and upper limits of *analytical* distributions (distributions from *samples* ignore this parameter and determine their limits based on the limits of the sample and the value of the `trim` parameter). E.g., if this is `c(.001, .999)`, then a slab is drawn for the distribution from the quantile at `p = .001` to the quantile at `p = .999`. If the lower (respectively upper) limit is `NA`, then the lower (upper) limit will be the minimum (maximum) of the distribution's support if it is finite, and `0.001` (`0.999`) if it is not finite. E.g., if `p_limits` is `c(NA, NA)`, on a gamma distribution the effective value of `p_limits` would be `c(0, .999)` since the gamma distribution is defined on `(0, Inf)`; whereas on a normal distribution it would be equivalent to `c(.001, .999)` since the normal distribution is defined on `(-Inf, Inf)`.

`density` <function | string> Density estimator for sample data. One of:

- A function which takes a numeric vector and returns a list with elements `x` (giving grid points for the density estimator) and `y` (the corresponding densities). **ggdist** provides a family of functions following this format, including `density_unbounded()` and `density_bounded()`. This format is also compatible with `stats::density()`.
- A string giving the suffix of a function name that starts with `"density_"`; e.g. `"bounded"` for `[density_bounded()]`, `"unbounded"` for `[density_unbounded()]`, or `"histogram"` for `density_histogram()`. Defaults to `"bounded"`, i.e. `density_bounded()`, which estimates the bounds from the data and then uses a bounded density estimator based on the reflection method.

adjust
: <scalar numeric | waiver> Passed to density (e.g. density_bounded()): Value to multiply the bandwidth of the density estimator by. Default waiver() defers to the default of the density estimator, which is usually 1.

trim
: <scalar logical | waiver> Passed to density (e.g. density_bounded()): Should the density estimate be trimmed to the range of the data? Default waiver() defers to the default of the density estimator, which is usually TRUE.

breaks
: <numeric | function | string | waiver> Passed to density (e.g. density_histogram()): Determines the breakpoints defining bins. Default waiver() defers to the default of the density estimator, which is usually "Scott". Similar to (but not exactly the same as) the breaks argument to graphics::hist(). One of:

- A scalar (length-1) numeric giving the number of bins
- A vector numeric giving the breakpoints between histogram bins
- A function taking x and weights and returning either the number of bins or a vector of breakpoints
- A string giving the suffix of a function that starts with "breaks_". **ggdist** provides weighted implementations of the "Sturges", "Scott", and "FD" break-finding algorithms from graphics::hist(), as well as breaks_fixed() for manually setting the bin width. See breaks.

For example, breaks = "Sturges" will use the breaks_Sturges() algorithm, breaks = 9 will create 9 bins, and breaks = breaks_fixed(width = 1) will set the bin width to 1.

align
: <scalar numeric | function | string | waiver> Passed to density (e.g. density_histogram()): Determines how to align the breakpoints defining bins. Default waiver() defers to the default of the density estimator, which is usually "none" (performs no alignment). One of:

- A scalar (length-1) numeric giving an offset that is subtracted from the breaks. The offset must be between 0 and the bin width.
- A function taking a sorted vector of breaks (bin edges) and returning an offset to subtract from the breaks.
- A string giving the suffix of a function that starts with "align_" used to determine the alignment, such as align_none(), align_boundary(), or align_center().

For example, align = "none" will provide no alignment, align = align_center(at = 0) will center a bin on 0, and align = align_boundary(at = 0) will align a bin edge on 0.

outline_bars
: <scalar logical | waiver> Passed to density (e.g. density_histogram()) and also used for discrete analytical distributions (whose slabs are drawn as histograms). Determines if outlines in between the bars are drawn. If waiver() or FALSE (the default), the outline is drawn only along the tops of the bars. If TRUE, outlines in between bars are also drawn (though you may have to set the slab_color or color aesthetic to see the outlines).

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

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

limits
: <length-2 numeric> 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
: <scalar numeric> Number of points at which to evaluate the function that defines the slab. Also passed to density (e.g. density_bounded()). Default waiver() uses the value 501 for analytical distributions and defers to the default of the density estimator for sample-based distributions, which is also usually 501.

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

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

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

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

na.rm
: <scalar logical> 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? 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). 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

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

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

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

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

Value

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

Computed Variables

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

- x or y: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is x or y depends on orientation
- xmin or ymin: For intervals, the lower end of the interval from the interval function.
- xmax or ymax: For intervals, the upper end of the interval from the interval function.
- .width: For intervals, the interval width as a numeric value in [0, 1]. For slabs, the width of the smallest interval containing that value of the slab.
- level: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.
- pdf: For slabs, the probability density function (PDF). If options("ggdist.experimental.slab_data_in_interval is TRUE: For intervals, the PDF at the point summary; intervals also have pdf_min and pdf_max for the PDF at the lower and upper ends of the interval.

- `cdf`: For slabs, the cumulative distribution function. If `options("ggdist.experimental.slab_data_in_intervals")` is `TRUE`: For intervals, the CDF at the point summary; intervals also have `cdf_min` and `cdf_max` for the CDF at the lower and upper ends of the interval.
- `n`: For slabs, the number of data points summarized into that slab. If the slab was created from an analytical distribution via the `xdist`, `ydist`, or `dist` aesthetic, `n` will be `Inf`.
- `f`: (deprecated) For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by `slab_type`. Instead of using `slab_type` to change `f` and then mapping `f` onto an aesthetic, it is now recommended to simply map the corresponding computed variable (e.g. `pdf`, `cdf`, or `1 - cdf`) directly onto the desired aesthetic.

Aesthetics

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

These `stats` support the following aesthetics:

- `x`: x position of the geometry (when orientation = `"vertical"`); or sample data to be summarized (when `orientation = "horizontal"` with sample data).
- `y`: y position of the geometry (when orientation = `"horizontal"`); or sample data to be summarized (when `orientation = "vertical"` with sample data).
- `weight`: When using samples (i.e. the `x` and `y` aesthetics, not `xdist` or `ydist`), optional weights to be applied to each draw.
- `xdist`: When using analytical distributions, distribution to map on the x axis: a **distributional** object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `ydist`: When using analytical distributions, distribution to map on the y axis: a **distributional** object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `dist`: When using analytical distributions, a name of a distribution (e.g. `"norm"`), a **distributional** object (e.g. `dist_normal()`), or a `posterior::rvar()` object. See **Details**.
- `args`: Distribution arguments (`args` or `arg1`, ... `arg9`). See **Details**.

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

Slab-specific aesthetics

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

- justification: Justification of the interval relative to the slab, where 0 indicates bottom/left justification and 1 indicates top/right justification (depending on orientation). If justification is NULL (the default), then it is set automatically based on the value of side: when side is "top"/"right" justification is set to 0, when side is "bottom"/"left" justification is set to 1, and when side is "both" justification is set to 0.5.
- datatype: When using composite geoms directly without a stat (e.g. geom_slabinterval()), datatype is used to indicate which part of the geom a row in the data targets: rows with datatype = "slab" target the slab portion of the geometry and rows with datatype = "interval" target the interval portion of the geometry. This is set automatically when using ggdist stats.

Interval-specific aesthetics

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

Point-specific aesthetics

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

Color aesthetics

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

Line aesthetics

- linewidth: Width of the line used to draw the **interval** (except with geom_slab(): then it is the width of the **slab**). With composite geometries including an interval and slab, use slab_linewidth to set the line width of the **slab** (see below). For **interval**, raw linewidth values are transformed according to the interval_size_domain and interval_size_range parameters of the geom (see above).
- size: Determines the size of the **point**. If linewidth is not provided, size will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only size and not linewidth). Raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the point_size aesthetic (below) to set sub-geometry size directly without applying the effects of interval_size_domain, interval_size_range, and fatten_point.

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

Slab-specific color and line override aesthetics

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

Interval-specific color and line override aesthetics

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

Point-specific color and line override aesthetics

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

Deprecated aesthetics

- slab_size: Use slab_linewidth.
- interval_size: Use interval_linewidth.

Other aesthetics (these work as in standard geoms)

- width
- height
- group

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

See Also

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

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

Examples

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

theme_set(theme_ggdist())

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

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

---

stat_gradientinterval *Gradient + interval plot (shortcut stat)*

Description

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

Roughly equivalent to:

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

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

Usage

```
stat_gradientinterval(
  mapping = NULL,
  data = NULL,
  geom = "slabinterval",
  position = "identity",
  ...,
  fill_type = "auto",
  p_limits = c(NA, NA),
  density = "bounded",
  adjust = waiver(),
  trim = waiver(),
  breaks = waiver(),
  align = waiver(),
  outline_bars = waiver(),
  expand = FALSE,
  point_interval = "median_qi",
  limits = NULL,
  n = waiver(),
  .width = c(0.66, 0.95),
  orientation = NA,
  na.rm = FALSE,
  show.legend = c(size = FALSE, slab_alpha = FALSE),
  inherit.aes = TRUE
)
```

Arguments

`mapping` Set of aesthetic mappings created by `aes()`. 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` <Geom | string> Use to override the default connection between `stat_gradientinterval()` and `geom_slabinterval()`

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

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

`subscale` <function | string> Sub-scale used to scale values of the `thickness` aesthetic within the groups determined by `normalize`. One of:

- A function that takes an `x` argument giving a numeric vector of values to be scaled and then returns a thickness vector representing the scaled values, such as `subscale_thickness()` or `subscale_identity()`.
- A string giving the name of such a function when prefixed with `"subscale_"`; e.g. `"thickness"` or `"identity"`. The value `"thickness"` using the default subscale, which can be modified by setting `subscale_thickness`; see the documentation for that function.

For a comprehensive discussion and examples of slab scaling and normalization, see the `thickness` scale article.

`normalize` <string> Groups within which to scale values of the `thickness` aesthetic. One of:

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

For a comprehensive discussion and examples of slab scaling and normalization, see the `thickness` scale article.

`interval_size_domain` <length-2 numeric> Minimum and maximum of the values of the `size` and `linewidth` aesthetics that will be translated into actual sizes for intervals drawn according to `interval_size_range` (see the documentation for that argument.)

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

`fatten_point` <scalar numeric> 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`.

`arrow` <arrow | NULL> Type of arrow heads to use on the interval, or `NULL` for no arrows.

`subguide` <function | string> Sub-guide used to annotate the `thickness` scale. One of:

- A function that takes a `scale` argument giving a ggplot2::Scale object and an `orientation` argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as `subguide_axis()` (to draw a traditional axis) or `subguide_none()` (to draw no annotation). See `subguide_axis()` for a list of possibilities and examples.
- A string giving the name of such a function when prefixed with `"subguide_"`; e.g. `"axis"` or `"none"`. The values `"slab"`, `"dots"`, and `"spike"` use the default subguide for their geom families (no subguide), which can be modified by setting `subguide_slab`, `subguide_dots`, or `subguide_spike`; see the documentation for those functions.

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

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

p_limits — <length-2 numeric> Probability limits. Used to determine the lower and upper limits of *analytical* distributions (distributions from *samples* ignore this parameter and determine their limits based on the limits of the sample and the value of the `trim` parameter). E.g., if this is `c(.001, .999)`, then a slab is drawn for the distribution from the quantile at `p = .001` to the quantile at `p = .999`. If the lower (respectively upper) limit is `NA`, then the lower (upper) limit will be the minimum (maximum) of the distribution's support if it is finite, and `0.001` (`0.999`) if it is not finite. E.g., if `p_limits` is `c(NA, NA)`, on a gamma distribution the effective value of `p_limits` would be `c(0, .999)` since the gamma distribution is defined on `(0, Inf)`; whereas on a normal distribution it would be equivalent to `c(.001, .999)` since the normal distribution is defined on `(-Inf, Inf)`.

density — <function | string> Density estimator for sample data. One of:

- A function which takes a numeric vector and returns a list with elements `x` (giving grid points for the density estimator) and `y` (the corresponding densities). **ggdist** provides a family of functions following this format, including `density_unbounded()` and `density_bounded()`. This format is also compatible with `stats::density()`.
- A string giving the suffix of a function name that starts with `"density_"`; e.g. `"bounded"` for `[density_bounded()]`, `"unbounded"` for `[density_unbounded()]`, or `"histogram"` for `density_histogram()`. Defaults to `"bounded"`, i.e. `density_bounded()`, which estimates the bounds from the data and then uses a bounded density estimator based on the reflection method.

adjust — <scalar numeric | waiver> Passed to `density` (e.g. `density_bounded()`): Value to multiply the bandwidth of the density estimator by. Default `waiver()` defers to the default of the density estimator, which is usually `1`.

trim — <scalar logical | waiver> Passed to `density` (e.g. `density_bounded()`): Should the density estimate be trimmed to the range of the data? Default `waiver()` defers to the default of the density estimator, which is usually `TRUE`.

breaks — <numeric | function | string | waiver> Passed to `density` (e.g. `density_histogram()`): Determines the breakpoints defining bins. Default `waiver()` defers to the default of the density estimator, which is usually `"Scott"`. Similar to (but not exactly the same as) the `breaks` argument to `graphics::hist()`. One of:

- A scalar (length-1) numeric giving the number of bins
- A vector numeric giving the breakpoints between histogram bins
- A function taking `x` and `weights` and returning either the number of bins or a vector of breakpoints
- A string giving the suffix of a function that starts with `"breaks_"`. **ggdist** provides weighted implementations of the `"Sturges"`, `"Scott"`, and `"FD"` break-finding algorithms from `graphics::hist()`, as well as `breaks_fixed()` for manually setting the bin width. See breaks.

For example, `breaks = "Sturges"` will use the `breaks_Sturges()` algorithm, `breaks = 9` will create 9 bins, and `breaks = breaks_fixed(width = 1)` will set the bin width to `1`.

align — <scalar numeric | function | string | waiver> Passed to `density` (e.g. `density_histogram()`): Determines how to align the breakpoints defining bins. Default `waiver()` defers to the default of the density estimator, which is usually `"none"` (performs no alignment). One of:

- A scalar (length-1) numeric giving an offset that is subtracted from the breaks. The offset must be between `0` and the bin width.
- A function taking a sorted vector of `breaks` (bin edges) and returning an offset to subtract from the breaks.
- A string giving the suffix of a function that starts with `"align_"` used to determine the alignment, such as `align_none()`, `align_boundary()`, or `align_center()`.

For example, `align = "none"` will provide no alignment, `align = align_center(at = 0)` will center a bin on `0`, and `align = align_boundary(at = 0)` will align a bin edge on `0`.

`outline_bars` <scalar logical | waiver> Passed to `density` (e.g. `density_histogram()`) and also used for discrete analytical distributions (whose slabs are drawn as histograms). Determines if outlines in between the bars are drawn. If `waiver()` or `FALSE` (the default), the outline is drawn only along the tops of the bars. If `TRUE`, outlines in between bars are also drawn (though you may have to set the `slab_color` or `color` aesthetic to see the outlines).

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

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

`limits` <length-2 numeric> 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` <scalar numeric> Number of points at which to evaluate the function that defines the slab. Also passed to `density` (e.g. `density_bounded()`). Default `waiver()` uses the value `501` for analytical distributions and defers to the default of the density estimator for sample-based distributions, which is also usually `501`.

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

`orientation` <string> Whether this geom is drawn horizontally or vertically. One of:

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

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

`na.rm` <scalar logical> 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? 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). 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

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

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

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

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

Value

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

Computed Variables

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

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

Aesthetics

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

These `stats` support the following aesthetics:

- `x`: x position of the geometry (when orientation = `"vertical"`); or sample data to be summarized (when `orientation = "horizontal"` with sample data).
- `y`: y position of the geometry (when orientation = `"horizontal"`); or sample data to be summarized (when `orientation = "vertical"` with sample data).
- `weight`: When using samples (i.e. the `x` and `y` aesthetics, not `xdist` or `ydist`), optional weights to be applied to each draw.
- `xdist`: When using analytical distributions, distribution to map on the x axis: a **distributional** object (e.g. `dist_normal()`) or a `posterior::rvar()` object.

- `ydist`: When using analytical distributions, distribution to map on the y axis: a **distributional** object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `dist`: When using analytical distributions, a name of a distribution (e.g. `"norm"`), a **distributional** object (e.g. `dist_normal()`), or a `posterior::rvar()` object. See **Details**.
- `args`: Distribution arguments (`args` or `arg1`, ... `arg9`). See **Details**.

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

Slab-specific aesthetics

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

Interval-specific aesthetics

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

Point-specific aesthetics

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

Color aesthetics

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

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

Line aesthetics

- linewidth: Width of the line used to draw the **interval** (except with geom_slab(): then it is the width of the **slab**). With composite geometries including an interval and slab, use slab_linewidth to set the line width of the **slab** (see below). For **interval**, raw linewidth values are transformed according to the interval_size_domain and interval_size_range parameters of the geom (see above).
- size: Determines the size of the **point**. If linewidth is not provided, size will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only size and not linewidth). Raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the point_size aesthetic (below) to set sub-geometry size directly without applying the effects of interval_size_domain, interval_size_range, and fatten_point.
- stroke: Width of the outline around the **point** sub-geometry.
- linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the slab_linetype or interval_linetype aesthetics (below) to set sub-geometry line types separately.

Slab-specific color and line override aesthetics

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

Interval-specific color and line override aesthetics

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

Point-specific color and line override aesthetics

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

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

Deprecated aesthetics

- `slab_size`: Use `slab_linewidth`.
- `interval_size`: Use `interval_linewidth`.

Other aesthetics (these work as in standard geoms)

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

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

See Also

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

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

Examples

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

theme_set(theme_ggdist())

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

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

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

---

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

Description

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

Usage

```
stat_halfeye(
  mapping = NULL,
  data = NULL,
  geom = "slabinterval",
  position = "identity",
  ...,
  p_limits = c(NA, NA),
  density = "bounded",
  adjust = waiver(),
  trim = waiver(),
  breaks = waiver(),
  align = waiver(),
  outline_bars = waiver(),
  expand = FALSE,
  point_interval = "median_qi",
  limits = NULL,
  n = waiver(),
  .width = c(0.66, 0.95),
  orientation = NA,
  na.rm = FALSE,
  show.legend = c(size = FALSE),
  inherit.aes = TRUE
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. 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
<Geom | string> Use to override the default connection between stat_halfeye() and geom_slabinterval()

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

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

subscale <function | string> Sub-scale used to scale values of the thickness aesthetic within the groups determined by normalize. One of:

- A function that takes an x argument giving a numeric vector of values to be scaled and then returns a thickness vector representing the scaled values, such as subscale_thickness() or subscale_identity().
- A string giving the name of such a function when prefixed with "subscale_"; e.g. "thickness" or "identity". The value "thickness" using the default subscale, which can be modified by setting subscale_thickness; see the documentation for that function.

For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.

normalize <string> Groups within which to scale values of the thickness aesthetic. One of:

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

For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.

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

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

interval_size_domain <length-2 numeric> Minimum and maximum of the values of the size and linewidth aesthetics that will be translated into actual sizes for intervals drawn according to interval_size_range (see the documentation for that argument.)

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

fatten_point <scalar numeric> 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.

arrow <arrow | NULL> Type of arrow heads to use on the interval, or NULL for no arrows.

subguide <function | string> Sub-guide used to annotate the thickness scale. One of:

- A function that takes a scale argument giving a ggplot2::Scale object and an orientation argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as subguide_axis() (to draw a traditional axis) or subguide_none() (to draw no annotation). See subguide_axis() for a list of possibilities and examples.
- A string giving the name of such a function when prefixed with "subguide_"; e.g. "axis" or "none". The values "slab", "dots", and "spike" use the default subguide for their geom families (no subguide), which can be modified by setting subguide_slab, subguide_dots, or subguide_spike; see the documentation for those functions.

p_limits <length-2 numeric> Probability limits. Used to determine the lower and upper limits of *analytical* distributions (distributions from *samples* ignore this parameter and determine their limits based on the limits of the sample and the value of the trim parameter). E.g., if this is c(.001, .999), then a slab is drawn for the distribution from the quantile at p = .001 to the quantile at p = .999. If the lower (respectively upper) limit is NA, then the lower (upper) limit will be the minimum (maximum) of the distribution's support if it is finite, and 0.001 (0.999) if it is not finite. E.g., if p_limits is c(NA, NA), on a gamma distribution the effective value of p_limits would be c(0, .999) since the gamma distribution is defined on (0, Inf); whereas on a normal distribution it would be equivalent to c(.001, .999) since the normal distribution is defined on (-Inf, Inf).

density <function | string> Density estimator for sample data. One of:

- A function which takes a numeric vector and returns a list with elements x (giving grid points for the density estimator) and y (the corresponding densities). **ggdist** provides a family of functions following this format, including density_unbounded() and density_bounded(). This format is also compatible with stats::density().
- A string giving the suffix of a function name that starts with "density_"; e.g. "bounded" for [density_bounded()], "unbounded" for [density_unbounded()], or "histogram" for density_histogram(). Defaults to "bounded", i.e. density_bounded(), which estimates the bounds from the data and then uses a bounded density estimator based on the reflection method.

adjust <scalar numeric | waiver> Passed to density (e.g. density_bounded()): Value to multiply the bandwidth of the density estimator by. Default waiver() defers to the default of the density estimator, which is usually 1.

trim <scalar logical | waiver> Passed to density (e.g. density_bounded()): Should the density estimate be trimmed to the range of the data? Default waiver() defers to the default of the density estimator, which is usually TRUE.

breaks <numeric | function | string | waiver> Passed to density (e.g. density_histogram()): Determines the breakpoints defining bins. Default waiver() defers to the default of the density estimator, which is usually "Scott". Similar to (but not exactly the same as) the breaks argument to graphics::hist(). One of:

- A scalar (length-1) numeric giving the number of bins

- A vector numeric giving the breakpoints between histogram bins
- A function taking x and weights and returning either the number of bins or a vector of breakpoints
- A string giving the suffix of a function that starts with "breaks_". **ggdist** provides weighted implementations of the "Sturges", "Scott", and "FD" break-finding algorithms from graphics::hist(), as well as breaks_fixed() for manually setting the bin width. See breaks.

For example, breaks = "Sturges" will use the breaks_Sturges() algorithm, breaks = 9 will create 9 bins, and breaks = breaks_fixed(width = 1) will set the bin width to 1.

align
<scalar numeric | function | string | waiver> Passed to density (e.g. density_histogram()): Determines how to align the breakpoints defining bins. Default waiver() defers to the default of the density estimator, which is usually "none" (performs no alignment). One of:

- A scalar (length-1) numeric giving an offset that is subtracted from the breaks. The offset must be between 0 and the bin width.
- A function taking a sorted vector of breaks (bin edges) and returning an offset to subtract from the breaks.
- A string giving the suffix of a function that starts with "align_" used to determine the alignment, such as align_none(), align_boundary(), or align_center().

For example, align = "none" will provide no alignment, align = align_center(at = 0) will center a bin on 0, and align = align_boundary(at = 0) will align a bin edge on 0.

outline_bars
<scalar logical | waiver> Passed to density (e.g. density_histogram()) and also used for discrete analytical distributions (whose slabs are drawn as histograms). Determines if outlines in between the bars are drawn. If waiver() or FALSE (the default), the outline is drawn only along the tops of the bars. If TRUE, outlines in between bars are also drawn (though you may have to set the slab_color or color aesthetic to see the outlines).

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

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

limits
<length-2 numeric> 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
: <scalar numeric> Number of points at which to evaluate the function that defines the slab. Also passed to density (e.g. density_bounded()). Default waiver() uses the value 501 for analytical distributions and defers to the default of the density estimator for sample-based distributions, which is also usually 501.

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

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

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

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

na.rm
: <scalar logical> 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? 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). 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

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

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

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

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

Value

A ggplot2::Stat representing a half-eye (density + interval) geometry which can be added to a ggplot() object.

Computed Variables

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

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

Aesthetics

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

These `stats` support the following aesthetics:

- `x`: x position of the geometry (when orientation = `"vertical"`); or sample data to be summarized (when `orientation` = `"horizontal"` with sample data).
- `y`: y position of the geometry (when orientation = `"horizontal"`); or sample data to be summarized (when `orientation` = `"vertical"` with sample data).
- `weight`: When using samples (i.e. the `x` and `y` aesthetics, not `xdist` or `ydist`), optional weights to be applied to each draw.
- `xdist`: When using analytical distributions, distribution to map on the x axis: a **distributional** object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `ydist`: When using analytical distributions, distribution to map on the y axis: a **distributional** object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `dist`: When using analytical distributions, a name of a distribution (e.g. `"norm"`), a **distributional** object (e.g. `dist_normal()`), or a `posterior::rvar()` object. See **Details**.
- `args`: Distribution arguments (`args` or `arg1`, ... `arg9`). See **Details**.

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

Slab-specific aesthetics

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

Interval-specific aesthetics

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

Point-specific aesthetics

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

Color aesthetics

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

Line aesthetics

- linewidth: Width of the line used to draw the **interval** (except with geom_slab(): then it is the width of the **slab**). With composite geometries including an interval and slab, use slab_linewidth to set the line width of the **slab** (see below). For **interval**, raw linewidth values are transformed according to the interval_size_domain and interval_size_range parameters of the geom (see above).
- size: Determines the size of the **point**. If linewidth is not provided, size will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only size and not linewidth). Raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the point_size aesthetic (below) to set sub-geometry size directly without applying the effects of interval_size_domain, interval_size_range, and fatten_point.
- stroke: Width of the outline around the **point** sub-geometry.
- linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the slab_linetype or interval_linetype aesthetics (below) to set sub-geometry line types separately.

Slab-specific color and line override aesthetics

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

- slab_alpha: Override for alpha: the opacity of the slab.
- slab_linewidth: Override for linwidth: the width of the outline of the slab.
- slab_linetype: Override for linetype: the line type of the outline of the slab.

Interval-specific color and line override aesthetics

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

Point-specific color and line override aesthetics

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

Deprecated aesthetics

- slab_size: Use slab_linewidth.
- interval_size: Use interval_linewidth.

Other aesthetics (these work as in standard geoms)

- width
- height
- group

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

See Also

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

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

Examples

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

theme_set(theme_ggdist())

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

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

---

stat_histinterval *Histogram + interval plot (shortcut stat)*

Description

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

Roughly equivalent to:

```
stat_slabinterval(
  density = "histogram"
)
```

Usage

```
stat_histinterval(
  mapping = NULL,
  data = NULL,
  geom = "slabinterval",
```

```
  position = "identity",
  ...,
  density = "histogram",
  p_limits = c(NA, NA),
  adjust = waiver(),
  trim = waiver(),
  breaks = waiver(),
  align = waiver(),
  outline_bars = waiver(),
  expand = FALSE,
  point_interval = "median_qi",
  limits = NULL,
  n = waiver(),
  .width = c(0.66, 0.95),
  orientation = NA,
  na.rm = FALSE,
  show.legend = c(size = FALSE),
  inherit.aes = TRUE
)
```

Arguments

`mapping` Set of aesthetic mappings created by `aes()`. 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` <Geom | string> Use to override the default connection between `stat_histinterval()` and `geom_slabinterval()`

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

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

`subscale` <function | string> Sub-scale used to scale values of the `thickness` aesthetic within the groups determined by `normalize`. One of:

- A function that takes an x argument giving a numeric vector of values to be scaled and then returns a thickness vector representing the scaled values, such as subscale_thickness() or subscale_identity().
- A string giving the name of such a function when prefixed with "subscale_"; e.g. "thickness" or "identity". The value "thickness" using the default subscale, which can be modified by setting subscale_thickness; see the documentation for that function.

For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.

normalize <string> Groups within which to scale values of the thickness aesthetic. One of:

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

For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.

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

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

interval_size_domain <length-2 numeric> Minimum and maximum of the values of the size and linewidth aesthetics that will be translated into actual sizes for intervals drawn according to interval_size_range (see the documentation for that argument.)

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

fatten_point <scalar numeric> 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.

arrow <arrow | NULL> Type of arrow heads to use on the interval, or NULL for no arrows.

subguide <function | string> Sub-guide used to annotate the thickness scale. One of:

- A function that takes a scale argument giving a ggplot2::Scale object and an orientation argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as subguide_axis() (to draw a traditional axis) or subguide_none() (to draw no annotation). See subguide_axis() for a list of possibilities and examples.
- A string giving the name of such a function when prefixed with "subguide_"; e.g. "axis" or "none". The values "slab", "dots", and "spike" use the default subguide for their geom families (no subguide), which can be modified by setting subguide_slab, subguide_dots, or subguide_spike; see the documentation for those functions.

density <function | string> Density estimator for sample data. One of:

- A function which takes a numeric vector and returns a list with elements x (giving grid points for the density estimator) and y (the corresponding densities). **ggdist** provides a family of functions following this format, including density_unbounded() and density_bounded(). This format is also compatible with stats::density().
- A string giving the suffix of a function name that starts with "density_"; e.g. "bounded" for [density_bounded()], "unbounded" for [density_unbounded()], or "histogram" for density_histogram(). Defaults to "bounded", i.e.

density_bounded(), which estimates the bounds from the data and then uses a bounded density estimator based on the reflection method.

p_limits
: <length-2 numeric> Probability limits. Used to determine the lower and upper limits of *analytical* distributions (distributions from *samples* ignore this parameter and determine their limits based on the limits of the sample and the value of the `trim` parameter). E.g., if this is `c(.001, .999)`, then a slab is drawn for the distribution from the quantile at `p = .001` to the quantile at `p = .999`. If the lower (respectively upper) limit is `NA`, then the lower (upper) limit will be the minimum (maximum) of the distribution's support if it is finite, and `0.001` (`0.999`) if it is not finite. E.g., if `p_limits` is `c(NA, NA)`, on a gamma distribution the effective value of `p_limits` would be `c(0, .999)` since the gamma distribution is defined on `(0, Inf)`; whereas on a normal distribution it would be equivalent to `c(.001, .999)` since the normal distribution is defined on `(-Inf, Inf)`.

adjust
: <scalar numeric | waiver> Passed to `density` (e.g. density_bounded()): Value to multiply the bandwidth of the density estimator by. Default waiver() defers to the default of the density estimator, which is usually `1`.

trim
: <scalar logical | waiver> Passed to `density` (e.g. density_bounded()): Should the density estimate be trimmed to the range of the data? Default waiver() defers to the default of the density estimator, which is usually `TRUE`.

breaks
: <numeric | function | string | waiver> Passed to `density` (e.g. density_histogram()): Determines the breakpoints defining bins. Default waiver() defers to the default of the density estimator, which is usually `"Scott"`. Similar to (but not exactly the same as) the `breaks` argument to graphics::hist(). One of:

- A scalar (length-1) numeric giving the number of bins
- A vector numeric giving the breakpoints between histogram bins
- A function taking `x` and `weights` and returning either the number of bins or a vector of breakpoints
- A string giving the suffix of a function that starts with `"breaks_"`. **ggdist** provides weighted implementations of the `"Sturges"`, `"Scott"`, and `"FD"` break-finding algorithms from graphics::hist(), as well as breaks_fixed() for manually setting the bin width. See breaks.

For example, `breaks = "Sturges"` will use the breaks_Sturges() algorithm, `breaks = 9` will create 9 bins, and `breaks = breaks_fixed(width = 1)` will set the bin width to `1`.

align
: <scalar numeric | function | string | waiver> Passed to `density` (e.g. density_histogram()): Determines how to align the breakpoints defining bins. Default waiver() defers to the default of the density estimator, which is usually `"none"` (performs no alignment). One of:

- A scalar (length-1) numeric giving an offset that is subtracted from the breaks. The offset must be between `0` and the bin width.
- A function taking a sorted vector of `breaks` (bin edges) and returning an offset to subtract from the breaks.
- A string giving the suffix of a function that starts with `"align_"` used to determine the alignment, such as align_none(), align_boundary(), or align_center().

For example, `align = "none"` will provide no alignment, `align = align_center(at = 0)` will center a bin on `0`, and `align = align_boundary(at = 0)` will align a bin edge on `0`.

`outline_bars` <scalar logical | waiver> Passed to `density` (e.g. `density_histogram()`) and also used for discrete analytical distributions (whose slabs are drawn as histograms). Determines if outlines in between the bars are drawn. If `waiver()` or `FALSE` (the default), the outline is drawn only along the tops of the bars. If `TRUE`, outlines in between bars are also drawn (though you may have to set the `slab_color` or `color` aesthetic to see the outlines).

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

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

`limits` <length-2 numeric> 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` <scalar numeric> Number of points at which to evaluate the function that defines the slab. Also passed to `density` (e.g. `density_bounded()`). Default `waiver()` uses the value `501` for analytical distributions and defers to the default of the density estimator for sample-based distributions, which is also usually `501`.

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

`orientation` <string> Whether this geom is drawn horizontally or vertically. One of:

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

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

`na.rm` <scalar logical> 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? 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). 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

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

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

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

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

Value

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

Computed Variables

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

- `x` or `y`: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is `x` or `y` depends on `orientation`

- `xmin` or `ymin`: For intervals, the lower end of the interval from the interval function.
- `xmax` or `ymax`: For intervals, the upper end of the interval from the interval function.
- `.width`: For intervals, the interval width as a numeric value in `[0, 1]`. For slabs, the width of the smallest interval containing that value of the slab.
- `level`: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.
- `pdf`: For slabs, the probability density function (PDF). If `options("ggdist.experimental.slab_data_in_interval` is `TRUE`: For intervals, the PDF at the point summary; intervals also have `pdf_min` and `pdf_max` for the PDF at the lower and upper ends of the interval.
- `cdf`: For slabs, the cumulative distribution function. If `options("ggdist.experimental.slab_data_in_intervals"` is `TRUE`: For intervals, the CDF at the point summary; intervals also have `cdf_min` and `cdf_max` for the CDF at the lower and upper ends of the interval.
- `n`: For slabs, the number of data points summarized into that slab. If the slab was created from an analytical distribution via the `xdist`, `ydist`, or `dist` aesthetic, `n` will be `Inf`.
- `f`: (deprecated) For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by `slab_type`. Instead of using `slab_type` to change `f` and then mapping `f` onto an aesthetic, it is now recommended to simply map the corresponding computed variable (e.g. `pdf`, `cdf`, or `1 - cdf`) directly onto the desired aesthetic.

Aesthetics

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

These `stats` support the following aesthetics:

- `x`: x position of the geometry (when orientation = `"vertical"`); or sample data to be summarized (when `orientation` = `"horizontal"` with sample data).
- `y`: y position of the geometry (when orientation = `"horizontal"`); or sample data to be summarized (when `orientation` = `"vertical"` with sample data).
- `weight`: When using samples (i.e. the `x` and `y` aesthetics, not `xdist` or `ydist`), optional weights to be applied to each draw.
- `xdist`: When using analytical distributions, distribution to map on the x axis: a **distributional** object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `ydist`: When using analytical distributions, distribution to map on the y axis: a **distributional** object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `dist`: When using analytical distributions, a name of a distribution (e.g. `"norm"`), a **distributional** object (e.g. `dist_normal()`), or a `posterior::rvar()` object. See **Details**.
- `args`: Distribution arguments (`args` or `arg1`, ... `arg9`). See **Details**.

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

Slab-specific aesthetics

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

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

Interval-specific aesthetics

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

Point-specific aesthetics

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

Color aesthetics

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

Line aesthetics

- linewidth: Width of the line used to draw the **interval** (except with geom_slab(): then it is the width of the **slab**). With composite geometries including an interval and slab, use slab_linewidth to set the line width of the **slab** (see below). For **interval**, raw linewidth values are transformed according to the interval_size_domain and interval_size_range parameters of the geom (see above).
- size: Determines the size of the **point**. If linewidth is not provided, size will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only size and not linewidth). Raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the point_size aesthetic (below) to set sub-geometry size directly without applying the effects of interval_size_domain, interval_size_range, and fatten_point.
- stroke: Width of the outline around the **point** sub-geometry.
- linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the slab_linetype or interval_linetype aesthetics (below) to set sub-geometry line types separately.

Slab-specific color and line override aesthetics

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

Interval-specific color and line override aesthetics

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

Point-specific color and line override aesthetics

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

Deprecated aesthetics

- slab_size: Use slab_linewidth.
- interval_size: Use interval_linewidth.

Other aesthetics (these work as in standard geoms)

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

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

See Also

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

Other slabinterval stats: `stat_ccdfinterval()`, `stat_cdfinterval()`, `stat_eye()`, `stat_gradientinterval()`, `stat_halfeye()`, `stat_interval()`, `stat_pointinterval()`, `stat_slab()`, `stat_spike()`

Examples

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

theme_set(theme_ggdist())

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

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

stat_interval *Multiple-interval plot (shortcut stat)*

Description

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

Roughly equivalent to:

```
stat_slabinterval(
  aes(
    colour = after_stat(level),
    size = NULL
  ),
  geom = "interval",
  show_point = FALSE,
  .width = c(0.5, 0.8, 0.95),
  show_slab = FALSE,
  show.legend = NA
)
```

Usage

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

Arguments

`mapping` Set of aesthetic mappings created by 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` <Geom | string> Use to override the default connection between `stat_interval()` and `geom_interval()`

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

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

- `interval_size_range` <length-2 numeric> This geom scales the raw size aesthetic values when drawing interval and point sizes, as they tend to be too thick when using the default settings of `scale_size_continuous()`, which give sizes with a range of `c(1, 6)`. The `interval_size_domain` value indicates the input domain of raw size values (typically this should be equal to the value of the `range` argument of the `scale_size_continuous()` function), and `interval_size_range` indicates the desired output range of the size values (the min and max of the actual sizes used to draw intervals). Most of the time it is not recommended to change the value of this argument, as it may result in strange scaling of legends; this argument is a holdover from earlier versions that did not have size aesthetics targeting the point and interval separately. If you want to adjust the size of the interval or points separately, you can also use the `linewidth` or `point_size` aesthetics; see sub-geometry-scales.
- `interval_size_domain` <length-2 numeric> Minimum and maximum of the values of the `size` and `linewidth` aesthetics that will be translated into actual sizes for intervals drawn according to `interval_size_range` (see the documentation for that argument.)
- `arrow` <arrow | NULL> Type of arrow heads to use on the interval, or `NULL` for no arrows.

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

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

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

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

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

na.rm
<scalar logical> 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? 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). 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

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

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

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

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

Value

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

Computed Variables

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

- x or y: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is x or y depends on orientation
- xmin or ymin: For intervals, the lower end of the interval from the interval function.
- xmax or ymax: For intervals, the upper end of the interval from the interval function.
- .width: For intervals, the interval width as a numeric value in [0, 1]. For slabs, the width of the smallest interval containing that value of the slab.
- level: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.
- pdf: For slabs, the probability density function (PDF). If options("ggdist.experimental.slab_data_in_interval is TRUE: For intervals, the PDF at the point summary; intervals also have pdf_min and pdf_max for the PDF at the lower and upper ends of the interval.
- cdf: For slabs, the cumulative distribution function. If options("ggdist.experimental.slab_data_in_intervals' is TRUE: For intervals, the CDF at the point summary; intervals also have cdf_min and cdf_max for the CDF at the lower and upper ends of the interval.

Aesthetics

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

These stats support the following aesthetics:

- x: x position of the geometry (when orientation = "vertical"); or sample data to be summarized (when orientation = "horizontal" with sample data).
- y: y position of the geometry (when orientation = "horizontal"); or sample data to be summarized (when orientation = "vertical" with sample data).
- weight: When using samples (i.e. the x and y aesthetics, not xdist or ydist), optional weights to be applied to each draw.
- xdist: When using analytical distributions, distribution to map on the x axis: a **distributional** object (e.g. dist_normal()) or a posterior::rvar() object.
- ydist: When using analytical distributions, distribution to map on the y axis: a **distributional** object (e.g. dist_normal()) or a posterior::rvar() object.
- dist: When using analytical distributions, a name of a distribution (e.g. "norm"), a **distributional** object (e.g. dist_normal()), or a posterior::rvar() object. See **Details**.
- args: Distribution arguments (args or arg1, ... arg9). See **Details**.

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

Interval-specific aesthetics

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

Color aesthetics

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

Line aesthetics

- `linewidth`: Width of the line used to draw the **interval** (except with `geom_slab()`: then it is the width of the **slab**). With composite geometries including an interval and slab, use `slab_linewidth` to set the line width of the **slab** (see below). For **interval**, raw `linewidth` values are transformed according to the `interval_size_domain` and `interval_size_range` parameters of the `geom` (see above).
- `size`: Determines the size of the **point**. If `linewidth` is not provided, `size` will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only `size` and not `linewidth`). Raw `size` values are transformed according to the `interval_size_domain`, `interval_size_range`, and `fatten_point` parameters of the `geom` (see above). Use the `point_size` aesthetic (below) to set sub-geometry size directly without applying the effects of `interval_size_domain`, `interval_size_range`, and `fatten_point`.
- `stroke`: Width of the outline around the **point** sub-geometry.
- `linetype`: Type of line (e.g., `"solid"`, `"dashed"`, etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the `slab_linetype` or `interval_linetype` aesthetics (below) to set sub-geometry line types separately.

Interval-specific color and line override aesthetics

- `interval_colour`: (or `interval_color`) Override for `colour`/`color`: the color of the interval.
- `interval_alpha`: Override for `alpha`: the opacity of the interval.

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

Deprecated aesthetics

- interval_size: Use interval_linewidth.

Other aesthetics (these work as in standard geoms)

- width
- height
- group

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

See Also

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

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

Examples

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

theme_set(theme_ggdist())

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

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

```
ggplot(aes(y = group, xdist = dist_normal(mean, sd))) +
stat_interval() +
scale_color_brewer()
```

---

stat_lineribbon *Line + multiple-ribbon plot (shortcut stat)*

Description

A combination of `stat_slabinterval()` and `geom_lineribbon()` with sensible defaults for making line + multiple-ribbon plots. While `geom_lineribbon()` is intended for use on data frames that have already been summarized using a `point_interval()` function, `stat_lineribbon()` is intended for use directly on data frames of draws or of analytical distributions, and will perform the summarization using a `point_interval()` function.

Roughly equivalent to:

```
stat_slabinterval(
  aes(
    group = after_stat(level),
    fill = after_stat(level),
    order = after_stat(level),
    size = NULL
  ),
  geom = "lineribbon",
  .width = c(0.5, 0.8, 0.95),
  show_slab = FALSE,
  show.legend = NA
)
```

Usage

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

Arguments

mapping
: Set of aesthetic mappings created by `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
: <Geom | string> Use to override the default connection between `stat_lineribbon()` and `geom_lineribbon()`

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

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

  `step` <scalar logical | string> Should the line/ribbon be drawn as a step function? One of:

  - `FALSE` (default): do not draw as a step function.
  - `"mid"` (or `TRUE`): draw steps midway between adjacent x values.
  - `"hv"`: draw horizontal-then-vertical steps.
  - `"vh"`: draw as vertical-then-horizontal steps.

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

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

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

`orientation` <string> Whether this geom is drawn horizontally or vertically. One of:

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

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

`na.rm` <scalar logical> 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

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

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

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

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

Value

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

Computed Variables

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

- `x` or `y`: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is `x` or `y` depends on `orientation`
- `xmin` or `ymin`: For intervals, the lower end of the interval from the interval function.
- `xmax` or `ymax`: For intervals, the upper end of the interval from the interval function.
- `.width`: For intervals, the interval width as a numeric value in `[0, 1]`. For slabs, the width of the smallest interval containing that value of the slab.
- `level`: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.
- `pdf`: For slabs, the probability density function (PDF). If `options("ggdist.experimental.slab_data_in_interval` is `TRUE`: For intervals, the PDF at the point summary; intervals also have `pdf_min` and `pdf_max` for the PDF at the lower and upper ends of the interval.
- `cdf`: For slabs, the cumulative distribution function. If `options("ggdist.experimental.slab_data_in_intervals'` is `TRUE`: For intervals, the CDF at the point summary; intervals also have `cdf_min` and `cdf_max` for the CDF at the lower and upper ends of the interval.

Aesthetics

The line+ribbon `stats` and `geoms` have a wide variety of aesthetics that control the appearance of their two sub-geometries: the **line** and the **ribbon**.

These `stats` support the following aesthetics:

- `x`: x position of the geometry (when orientation = `"vertical"`); or sample data to be summarized (when `orientation` = `"horizontal"` with sample data).
- `y`: y position of the geometry (when orientation = `"horizontal"`); or sample data to be summarized (when `orientation` = `"vertical"` with sample data).
- `weight`: When using samples (i.e. the `x` and `y` aesthetics, not `xdist` or `ydist`), optional weights to be applied to each draw.
- `xdist`: When using analytical distributions, distribution to map on the x axis: a **distributional** object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `ydist`: When using analytical distributions, distribution to map on the y axis: a **distributional** object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `dist`: When using analytical distributions, a name of a distribution (e.g. `"norm"`), a **distributional** object (e.g. `dist_normal()`), or a `posterior::rvar()` object. See **Details**.
- `args`: Distribution arguments (`args` or `arg1`, ... `arg9`). See **Details**.

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

Ribbon-specific aesthetics

- `xmin`: Left edge of the ribbon sub-geometry (if `orientation` = `"horizontal"`).

- `xmax`: Right edge of the ribbon sub-geometry (if `orientation = "horizontal"`).
- `ymin`: Lower edge of the ribbon sub-geometry (if `orientation = "vertical"`).
- `ymax`: Upper edge of the ribbon sub-geometry (if `orientation = "vertical"`).
- `order`: The order in which ribbons are drawn. Ribbons with the smallest mean value of `order` are drawn first (i.e., will be drawn below ribbons with larger mean values of `order`). If `order` is not supplied to `geom_lineribbon()`, `-abs(xmax - xmin)` or `-abs(ymax - ymax)` (depending on `orientation`) is used, having the effect of drawing the widest (on average) ribbons on the bottom. `stat_lineribbon()` uses `order = after_stat(level)` by default, causing the ribbons generated from the largest `.width` to be drawn on the bottom.

Color aesthetics

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

Line aesthetics

- `linewidth`: Width of **line**. In **ggplot2** < 3.4, was called `size`.
- `linetype`: Type of **line** (e.g., `"solid"`, `"dashed"`, etc)

Other aesthetics (these work as in standard geoms)

- `group`

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

See Also

See `geom_lineribbon()` for the geom underlying this stat.

Other lineribbon stats: `stat_ribbon()`

Examples

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

theme_set(theme_ggdist())

# ON SAMPLE DATA
set.seed(12345)
tibble(
  x = rep(1:10, 100),
  y = rnorm(1000, x)
```

```
) %>%
  ggplot(aes(x = x, y = y)) +
  stat_lineribbon() +
  scale_fill_brewer()

# ON ANALYTICAL DISTRIBUTIONS
# Vectorized distribution types, like distributional::dist_normal()
# and posterior::rvar(), can be used with the `xdist` / `ydist` aesthetics
tibble(
  x = 1:10,
  sd = seq(1, 3, length.out = 10)
) %>%
  ggplot(aes(x = x, ydist = dist_normal(x, sd))) +
  stat_lineribbon() +
  scale_fill_brewer()
```

---

stat_mcse_dots *Blurry MCSE dot plot (stat)*

Description

Variant of `stat_dots()` for creating blurry dotplots of quantiles. Uses `posterior::mcse_quantile()` to calculate the Monte Carlo Standard Error of each quantile computed for the dotplot, yielding an `se` computed variable that is by default mapped onto the `sd` aesthetic of `geom_blur_dots()`.

Usage

```
stat_mcse_dots(
  mapping = NULL,
  data = NULL,
  geom = "blur_dots",
  position = "identity",
  ...,
  quantiles = NA,
  orientation = NA,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping` Set of aesthetic mappings created by `aes()`. 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` <Geom | string> Use to override the default connection between `stat_mcse_dots()` and `geom_blur_dots()`

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

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

`blur` <function | string> Blur function to apply to dots. One of:

- A function that takes a numeric vector of distances from the dot center, the dot radius, and the standard deviation of the blur and returns a vector of opacities in $[0, 1]$, such as `blur_gaussian()` or `blur_interval()`.
- A string indicating what blur function to use, as the suffix to a function name starting with `blur_`; e.g. `"gaussian"` (the default) applies `blur_gaussian()`.

`binwidth` <numeric | unit> The bin width to use for laying out the dots. One of:

- `NA` (the default): Dynamically select the bin width based on the size of the plot when drawn. This will pick a `binwidth` such that the tallest stack of dots is at most `scale` in height (ideally exactly `scale` in height, though this is not guaranteed).
- A length-1 (scalar) numeric or unit object giving the exact bin width.
- A length-2 (vector) numeric or unit object giving the minimum and maximum desired bin width. The bin width will be dynamically selected within these bounds.

If the value is numeric, it is assumed to be in units of data. The bin width (or its bounds) can also be specified using `unit()`, which may be useful if it is desired that the dots be a certain point size or a certain percentage of the width/height of the viewport. For example, `unit(0.1, "npc")` would make dots that are *exactly* 10% of the viewport size along whichever dimension the dotplot is drawn; `unit(c(0, 0.1), "npc")` would make dots that are *at most* 10% of the viewport size (while still ensuring the tallest stack is less than or equal to `scale`).

`dotsize` <scalar numeric> The width of the dots relative to the `binwidth`. The default, `1.07`, makes dots be just a bit wider than the bin width, which is a manually-tuned parameter that tends to work well with the default circular shape, preventing gaps between bins from appearing to be too large visually (as might arise from dots being *precisely* the `binwidth`). If it is desired to have dots be precisely the `binwidth`, set `dotsize = 1`.

stackratio <scalar numeric> The distance between the center of the dots in the same stack relative to the dot height. The default, 1, makes dots in the same stack just touch each other.

layout <string> The layout method used for the dots. One of:

- "bin" (default): places dots on the off-axis at the midpoint of their bins as in the classic Wilkinson dotplot. This maintains the alignment of rows and columns in the dotplot. This layout is slightly different from the classic Wilkinson algorithm in that: (1) it nudges bins slightly to avoid overlapping bins and (2) if the input data are symmetrical it will return a symmetrical layout.
- "weave": uses the same basic binning approach of "bin", but places dots in the off-axis at their actual positions (unless overlaps = "nudge", in which case overlaps may be nudged out of the way). This maintains the alignment of rows but does not align dots within columns.
- "hex": uses the same basic binning approach of "bin", but alternates placing dots + binwidth/4 or - binwidth/4 in the off-axis from the bin center. This allows hexagonal packing by setting a stackratio less than 1 (something like 0.9 tends to work).
- "swarm": uses the "compactswarm" layout from beeswarm::beeswarm(). Does not maintain alignment of rows or columns, but can be more compact and neat looking, especially for sample data (as opposed to quantile dotplots of theoretical distributions, which may look better with "bin", "weave", or "hex").
- "bar": for discrete distributions, lays out duplicate values in rectangular bars.

overlaps <string> How to handle overlapping dots or bins in the "bin", "weave", and "hex" layouts (dots never overlap in the "swarm" or "bar" layouts). For the purposes of this argument, dots are only considered to be overlapping if they would be overlapping when dotsize = 1 and stackratio = 1; i.e. if you set those arguments to other values, overlaps may still occur. One of:

- "keep": leave overlapping dots as they are. Dots may overlap (usually only slightly) in the "bin", "weave", and "hex" layouts.
- "nudge": nudge overlapping dots out of the way. Overlaps are avoided using a constrained optimization which minimizes the squared distance of dots to their desired positions, subject to the constraint that adjacent dots do not overlap.

smooth <function | string> Smoother to apply to dot positions. One of:

- A function that takes a numeric vector of dot positions and returns a smoothed version of that vector, such as smooth_bounded(), smooth_unbounded(), smooth_discrete(), or smooth_bar()‘.
- A string indicating what smoother to use, as the suffix to a function name starting with smooth_; e.g. "none" (the default) applies smooth_none(), which simply returns the given vector without applying smoothing.

Smoothing is most effective when the smoother is matched to the support of the distribution; e.g. using smooth_bounded(bounds = ...).

overflow <string> How to handle overflow of dots beyond the extent of the geom when a minimum binwidth (or an exact binwidth) is supplied. One of:

- "keep": Keep the overflow, drawing dots outside the geom bounds.
- "warn": Keep the overflow, but produce a warning suggesting solutions, such as setting binwidth = NA or overflow = "compress".
- "compress": Compress the layout. Reduces the binwidth to the size necessary to keep the dots within bounds, then adjusts stackratio and dotsize so that the apparent dot size is the user-specified minimum binwidth times the user-specified dotsize.

If you find the default layout has dots that are too small, and you are okay with dots overlapping, consider setting overflow = "compress" and supplying an exact or minimum dot size using binwidth.

verbose <scalar logical> If TRUE, print out the bin width of the dotplot. Can be useful if you want to start from an automatically-selected bin width and then adjust it manually. Bin width is printed both as data units and as normalized parent coordinates or "npc"s (see unit()). Note that if you just want to scale the selected bin width to fit within a desired area, it is probably easier to use scale than to copy and scale binwidth manually, and if you just want to provide constraints on the bin width, you can pass a length-2 vector to binwidth.

subguide <function | string> Sub-guide used to annotate the thickness scale. One of:

- A function that takes a scale argument giving a ggplot2::Scale object and an orientation argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as subguide_axis() (to draw a traditional axis) or subguide_none() (to draw no annotation). See subguide_axis() for a list of possibilities and examples.
- A string giving the name of such a function when prefixed with "subguide_"; e.g. "axis" or "none". The values "slab", "dots", and "spike" use the default subguide for their geom families (no subguide), which can be modified by setting subguide_slab, subguide_dots, or subguide_spike; see the documentation for those functions.

quantiles <scalar logical> Number of quantiles to plot in the dotplot. Use NA (the default) to plot all data points. Setting this to a value other than NA will produce a quantile dotplot: that is, a dotplot of quantiles from the sample or distribution (for analytical distributions, the default of NA is taken to mean 100 quantiles). See Kay et al. (2016) and Fernandes et al. (2018) for more information on quantile dotplots.

orientation <string> Whether this geom is drawn horizontally or vertically. One of:

- NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
- "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify different groups. For each group, uses the x, xmin, xmax, and thickness aesthetics to draw points, intervals, and slabs.

	• "vertical" (or "x"): draw vertically, using the x aesthetic to identify different groups. For each group, uses the y, ymin, ymax, and thickness aesthetics to draw points, intervals, and slabs. For compatibility with the base ggplot naming scheme for orientation, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal" (**ggdist** had an orientation parameter before base ggplot did, hence the discrepancy).
na.rm	<scalar logical> 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

The *dots* family of stats and geoms are similar to ggplot2::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 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)

Stats and geoms in this family include:

- geom_dots(): dotplots on raw data. Ensures the dotplot fits within available space by reducing the size of the dots automatically (may result in very small dots).
- geom_swarm() and geom_weave(): dotplots on raw data with defaults intended to create "beeswarm" plots. Used side = "both" by default, and sets the default dot size to the same size as geom_point() (binwidth = unit(1.5, "mm")), allowing dots to overlap instead of getting very small.
- stat_dots(): dotplots on raw data, **distributional** objects, and posterior::rvar()s
- geom_dotsinterval(): dotplot + interval plots on raw data with already-calculated intervals (rarely useful directly).
- stat_dotsinterval(): dotplot + interval plots on raw data, **distributional** objects, and posterior::rvar()s (will calculate intervals for you).

- geom_blur_dots(): blurry dotplots that allow the standard deviation of a blur applied to each dot to be specified using the sd aesthetic.
- stat_mcse_dots(): blurry dotplots of quantiles using the Monte Carlo Standard Error of each quantile.

stat_dots() and stat_dotsinterval(), when used with the quantiles argument, are particularly useful for constructing quantile dotplots, which can be an effective way to communicate uncertainty using a frequency framing that may be easier for laypeople to understand (Kay et al. 2016, Fernandes et al. 2018).

To visualize sample data, such as a data distribution, samples from a bootstrap distribution, or a Bayesian posterior, you can supply samples to the x or y aesthetic.

To visualize analytical distributions, you can use the xdist or ydist aesthetic. For historical reasons, you can also use dist to specify the distribution, though this is not recommended as it does not work as well with orientation detection. These aesthetics can be used as follows:

- xdist, ydist, and dist can be any distribution object from the distributional package (dist_normal(), dist_beta(), etc) or can be a posterior::rvar() object. Since these functions are vectorized, other columns can be passed directly to them in an aes() specification; e.g. aes(dist = dist_normal(mu, sigma)) will work if mu and sigma are columns in the input data frame.
- dist can be a character vector giving the distribution name. Then the arg1, ... arg9 aesthetics (or args as a list column) specify distribution arguments. Distribution names should correspond to R functions that have "p", "q", and "d" functions; e.g. "norm" is a valid distribution name because R defines the pnorm(), qnorm(), and dnorm() functions for Normal distributions.

  See the parse_dist() function for a useful way to generate dist and args values from human-readable distribution specs (like "normal(0,1)"). Such specs are also produced by other packages (like the brms::get_prior function in brms); thus, parse_dist() combined with the stats described here can help you visualize the output of those functions.

Value

A ggplot2::Stat representing a blurry MCSE dot geometry which can be added to a ggplot() object.

Computed Variables

The following variables are computed by this stat and made available for use in aesthetic specifications (aes()) using the after_stat() function or the after_stat argument of stage():

- x or y: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is x or y depends on orientation
- xmin or ymin: For intervals, the lower end of the interval from the interval function.
- xmax or ymax: For intervals, the upper end of the interval from the interval function.
- .width: For intervals, the interval width as a numeric value in [0, 1]. For slabs, the width of the smallest interval containing that value of the slab.
- level: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.

- pdf: For slabs, the probability density function (PDF). If options("ggdist.experimental.slab_data_in_intervals") is TRUE: For intervals, the PDF at the point summary; intervals also have pdf_min and pdf_max for the PDF at the lower and upper ends of the interval.
- cdf: For slabs, the cumulative distribution function. If options("ggdist.experimental.slab_data_in_intervals") is TRUE: For intervals, the CDF at the point summary; intervals also have cdf_min and cdf_max for the CDF at the lower and upper ends of the interval.
- n: For slabs, the number of data points summarized into that slab. If the slab was created from an analytical distribution via the xdist, ydist, or dist aesthetic, n will be Inf.
- f: (deprecated) For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by slab_type. Instead of using slab_type to change f and then mapping f onto an aesthetic, it is now recommended to simply map the corresponding computed variable (e.g. pdf, cdf, or 1 - cdf) directly onto the desired aesthetic.
- se: For dots, the Monte Carlo Standard Error of the quantile corresponding to each dot.

Aesthetics

The dots+interval stats and geoms have a wide variety of aesthetics that control the appearance of their three sub-geometries: the **dots** (aka the **slab**), the **point**, and the **interval**.

These stats support the following aesthetics:

- x: x position of the geometry (when orientation = "vertical"); or sample data to be summarized (when orientation = "horizontal" with sample data).
- y: y position of the geometry (when orientation = "horizontal"); or sample data to be summarized (when orientation = "vertical" with sample data).
- weight: When using samples (i.e. the x and y aesthetics, not xdist or ydist), optional weights to be applied to each draw.
- xdist: When using analytical distributions, distribution to map on the x axis: a **distributional** object (e.g. dist_normal()) or a posterior::rvar() object.
- ydist: When using analytical distributions, distribution to map on the y axis: a **distributional** object (e.g. dist_normal()) or a posterior::rvar() object.
- dist: When using analytical distributions, a name of a distribution (e.g. "norm"), a **distributional** object (e.g. dist_normal()), or a posterior::rvar() object. See **Details**.
- args: Distribution arguments (args or arg1, ... arg9). See **Details**.

In addition, in their default configuration (paired with geom_blur_dots()) the following aesthetics are supported by the underlying geom:

Dots-specific (aka Slab-specific) aesthetics

- sd: The standard deviation (in data units) of the blur associated with each dot.
- order: The order in which data points are stacked within bins. Can be used to create the effect of "stacked" dots by ordering dots according to a discrete variable. If omitted (NULL), the value of the data points themselves are used to determine stacking order. Only applies when layout is "bin" or "hex", as the other layout methods fully determine both *x* and *y* positions.

- `side`: Which side to place the slab on. `"topright"`, `"top"`, and `"right"` are synonyms which cause the slab to be drawn on the top or the right depending on if `orientation` is `"horizontal"` or `"vertical"`. `"bottomleft"`, `"bottom"`, and `"left"` are synonyms which cause the slab to be drawn on the bottom or the left depending on if `orientation` is `"horizontal"` or `"vertical"`. `"topleft"` causes the slab to be drawn on the top or the left, and `"bottomright"` causes the slab to be drawn on the bottom or the right. `"both"` draws the slab mirrored on both sides (as in a violin plot).
- `scale`: What proportion of the region allocated to this geom to use to draw the slab. If `scale = 1`, slabs that use the maximum range will just touch each other. Default is `0.9` to leave some space between adjacent slabs. For a comprehensive discussion and examples of slab scaling and normalization, see the `thickness` scale article.
- `justification`: Justification of the interval relative to the slab, where `0` indicates bottom/left justification and `1` indicates top/right justification (depending on `orientation`). If `justification` is `NULL` (the default), then it is set automatically based on the value of `side`: when `side` is `"top"/"right"` `justification` is set to `0`, when `side` is `"bottom"/"left"` `justification` is set to `1`, and when `side` is `"both"` `justification` is set to 0.5.
- `datatype`: When using composite geoms directly without a `stat` (e.g. `geom_slabinterval()`), `datatype` is used to indicate which part of the geom a row in the data targets: rows with `datatype = "slab"` target the slab portion of the geometry and rows with `datatype = "interval"` target the interval portion of the geometry. This is set automatically when using ggdist `stats`.

Interval-specific aesthetics

- `xmin`: Left end of the interval sub-geometry (if `orientation = "horizontal"`).
- `xmax`: Right end of the interval sub-geometry (if `orientation = "horizontal"`).
- `ymin`: Lower end of the interval sub-geometry (if `orientation = "vertical"`).
- `ymax`: Upper end of the interval sub-geometry (if `orientation = "vertical"`).

Point-specific aesthetics

- `shape`: Shape type used to draw the **point** sub-geometry.

Color aesthetics

- `colour`: (or `color`) The color of the **interval** and **point** sub-geometries. Use the `slab_color`, `interval_color`, or `point_color` aesthetics (below) to set sub-geometry colors separately.
- `fill`: The fill color of the **slab** and **point** sub-geometries. Use the `slab_fill` or `point_fill` aesthetics (below) to set sub-geometry colors separately.
- `alpha`: The opacity of the **slab**, **interval**, and **point** sub-geometries. Use the `slab_alpha`, `interval_alpha`, or `point_alpha` aesthetics (below) to set sub-geometry colors separately.
- `colour_ramp`: (or `color_ramp`) A secondary scale that modifies the `color` scale to "ramp" to another color. See `scale_colour_ramp()` for examples.
- `fill_ramp`: A secondary scale that modifies the `fill` scale to "ramp" to another color. See `scale_fill_ramp()` for examples.

Line aesthetics

- `linewidth`: Width of the line used to draw the **interval** (except with `geom_slab()`: then it is the width of the **slab**). With composite geometries including an interval and slab, use `slab_linewidth` to set the line width of the **slab** (see below). For **interval**, raw `linewidth` values are transformed according to the `interval_size_domain` and `interval_size_range` parameters of the `geom` (see above).
- `size`: Determines the size of the **point**. If `linewidth` is not provided, `size` will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only `size` and not `linewidth`). Raw `size` values are transformed according to the `interval_size_domain`, `interval_size_range`, and `fatten_point` parameters of the `geom` (see above). Use the `point_size` aesthetic (below) to set sub-geometry size directly without applying the effects of `interval_size_domain`, `interval_size_range`, and `fatten_point`.
- `stroke`: Width of the outline around the **point** sub-geometry.
- `linetype`: Type of line (e.g., `"solid"`, `"dashed"`, etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the `slab_linetype` or `interval_linetype` aesthetics (below) to set sub-geometry line types separately.

Slab-specific color and line override aesthetics

- `slab_fill`: Override for `fill`: the fill color of the slab.
- `slab_colour`: (or `slab_color`) Override for `colour`/`color`: the outline color of the slab.
- `slab_alpha`: Override for `alpha`: the opacity of the slab.
- `slab_linewidth`: Override for `linwidth`: the width of the outline of the slab.
- `slab_linetype`: Override for `linetype`: the line type of the outline of the slab.
- `slab_shape`: Override for `shape`: the shape of the dots used to draw the dotplot slab.

Interval-specific color and line override aesthetics

- `interval_colour`: (or `interval_color`) Override for `colour`/`color`: the color of the interval.
- `interval_alpha`: Override for `alpha`: the opacity of the interval.
- `interval_linetype`: Override for `linetype`: the line type of the interval.

Point-specific color and line override aesthetics

- `point_fill`: Override for `fill`: the fill color of the point.
- `point_colour`: (or `point_color`) Override for `colour`/`color`: the outline color of the point.
- `point_alpha`: Override for `alpha`: the opacity of the point.
- `point_size`: Override for `size`: the size of the point.

Deprecated aesthetics

- `slab_size`: Use `slab_linewidth`.
- `interval_size`: Use `interval_linewidth`.

Other aesthetics (these work as in standard `geom`s)

- width
- height
- group

See examples of some of these aesthetics in action in vignette("dotsinterval"). Learn more about the sub-geom override aesthetics (like interval_color) in the scales documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").

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 geom_blur_dots() for the geom underlying this stat. See vignette("dotsinterval") for a variety of examples of use.

Other dotsinterval stats: stat_dots(), stat_dotsinterval()

Examples

```
library(dplyr)
library(ggplot2)

theme_set(theme_ggdist())

set.seed(1234)
data.frame(x = rnorm(1000)) %>%
  ggplot(aes(x = x)) +
  stat_mcse_dots(quantiles = 100, layout = "weave")
```

---

stat_pointinterval *Point + multiple-interval plot (shortcut stat)*

Description

Shortcut version of stat_slabinterval() with geom_pointinterval() for creating point + multiple-interval plots.

Roughly equivalent to:

```
stat_slabinterval(
  geom = "pointinterval",
  show_slab = FALSE
)
```

Usage

```
stat_pointinterval(
  mapping = NULL,
  data = NULL,
  geom = "pointinterval",
  position = "identity",
  ...,
  point_interval = "median_qi",
  .width = c(0.66, 0.95),
  orientation = NA,
  na.rm = FALSE,
  show.legend = c(size = FALSE),
  inherit.aes = TRUE
)
```

Arguments

mapping
: Set of aesthetic mappings created by aes(). 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
: <Geom | string> Use to override the default connection between stat_pointinterval() and geom_pointinterval()

position
: <Position | string> Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to "dodge" (position_dodge()) or "dodgejust" (position_dodgejust()) can be useful if you have overlapping geometries.

...
: Other arguments passed to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or linewidth = 3 (see **Aesthetics**, below). They may also be parameters to the paired geom/stat. When paired with the default geom, geom_pointinterval(), these include:

  interval_size_domain <length-2 numeric> Minimum and maximum of the values of the size and linewidth aesthetics that will be translated into actual sizes for intervals drawn according to interval_size_range (see the documentation for that argument.)

  interval_size_range <length-2 numeric> This geom scales the raw size aesthetic values when drawing interval and point sizes, as they tend to be too thick when using the default settings of scale_size_continuous(), which give sizes with a range of `c(1, 6)`. The `interval_size_domain` value indicates the input domain of raw size values (typically this should be equal to the value of the `range` argument of the `scale_size_continuous()` function), and `interval_size_range` indicates the desired output range of the size values (the min and max of the actual sizes used to draw intervals). Most of the time it is not recommended to change the value of this argument, as it may result in strange scaling of legends; this argument is a holdover from earlier versions that did not have size aesthetics targeting the point and interval separately. If you want to adjust the size of the interval or points separately, you can also use the `linewidth` or `point_size` aesthetics; see sub-geometry-scales.

`fatten_point` <scalar numeric> 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`.

`arrow` <arrow | NULL> Type of arrow heads to use on the interval, or `NULL` for no arrows.

`point_interval` <function | string> A function from the `point_interval()` family (e.g., `median_qi`, `mean_qi`, `mode_hdi`, etc), or a string giving the name of a function from that family (e.g., `"median_qi"`, `"mean_qi"`, `"mode_hdi"`, etc; if a string, the caller's environment is searched for the function, followed by the **ggdist** environment). This function determines the point summary (typically mean, median, or mode) and interval type (quantile interval, `qi`; highest-density interval, `hdi`; or highest-density continuous interval, `hdci`). Output will be converted to the appropriate `x`- or `y`-based aesthetics depending on the value of `orientation`. See the `point_interval()` family of functions for more information.

`.width` <numeric> The `.width` argument passed to `point_interval`: a vector of probabilities to use that determine the widths of the resulting intervals. If multiple probabilities are provided, multiple intervals per group are generated, each with a different probability interval (and value of the corresponding `.width` and `level` generated variables).

`orientation` <string> Whether this geom is drawn horizontally or vertically. One of:

- `NA` (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
- `"horizontal"` (or `"y"`): draw horizontally, using the `y` aesthetic to identify different groups. For each group, uses the `x`, `xmin`, `xmax`, and `thickness` aesthetics to draw points, intervals, and slabs.
- `"vertical"` (or `"x"`): draw vertically, using the `x` aesthetic to identify different groups. For each group, uses the `y`, `ymin`, `ymax`, and `thickness` aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for `orientation`, `"x"` can be used as an alias for `"vertical"` and `"y"` as an alias for `"horizontal"` (**ggdist** had an `orientation` parameter before base ggplot did, hence the discrepancy).

`na.rm` <scalar logical> 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? 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). 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

To visualize sample data, such as a data distribution, samples from a bootstrap distribution, or a Bayesian posterior, you can supply samples to the `x` or `y` aesthetic.

To visualize analytical distributions, you can use the `xdist` or `ydist` aesthetic. For historical reasons, you can also use `dist` to specify the distribution, though this is not recommended as it does not work as well with orientation detection. These aesthetics can be used as follows:

- `xdist`, `ydist`, and `dist` can be any distribution object from the distributional package (`dist_normal()`, `dist_beta()`, etc) or can be a `posterior::rvar()` object. Since these functions are vectorized, other columns can be passed directly to them in an `aes()` specification; e.g. `aes(dist = dist_normal(mu, sigma))` will work if `mu` and `sigma` are columns in the input data frame.
- `dist` can be a character vector giving the distribution name. Then the `arg1`, ... `arg9` aesthetics (or `args` as a list column) specify distribution arguments. Distribution names should correspond to R functions that have `"p"`, `"q"`, and `"d"` functions; e.g. `"norm"` is a valid distribution name because R defines the `pnorm()`, `qnorm()`, and `dnorm()` functions for Normal distributions.

  See the `parse_dist()` function for a useful way to generate `dist` and `args` values from human-readable distribution specs (like `"normal(0,1)"`). Such specs are also produced by other packages (like the `brms::get_prior` function in brms); thus, `parse_dist()` combined with the stats described here can help you visualize the output of those functions.

Value

A ggplot2::Stat representing a point + multiple-interval geometry which can be added to a `ggplot()` object.

Computed Variables

The following variables are computed by this stat and made available for use in aesthetic specifications (`aes()`) using the `after_stat()` function or the `after_stat` argument of `stage()`:

- `x` or `y`: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is `x` or `y` depends on `orientation`
- `xmin` or `ymin`: For intervals, the lower end of the interval from the interval function.
- `xmax` or `ymax`: For intervals, the upper end of the interval from the interval function.
- `.width`: For intervals, the interval width as a numeric value in `[0, 1]`. For slabs, the width of the smallest interval containing that value of the slab.

- `level`: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.
- `pdf`: For slabs, the probability density function (PDF). If `options("ggdist.experimental.slab_data_in_interval` is `TRUE`: For intervals, the PDF at the point summary; intervals also have `pdf_min` and `pdf_max` for the PDF at the lower and upper ends of the interval.
- `cdf`: For slabs, the cumulative distribution function. If `options("ggdist.experimental.slab_data_in_intervals'` is `TRUE`: For intervals, the CDF at the point summary; intervals also have `cdf_min` and `cdf_max` for the CDF at the lower and upper ends of the interval.

Aesthetics

The slab+interval `stats` and `geoms` have a wide variety of aesthetics that control the appearance of their three sub-geometries: the **slab**, the **point**, and the **interval**.

These `stats` support the following aesthetics:

- `x`: x position of the geometry (when orientation = `"vertical"`); or sample data to be summarized (when `orientation = "horizontal"` with sample data).
- `y`: y position of the geometry (when orientation = `"horizontal"`); or sample data to be summarized (when `orientation = "vertical"` with sample data).
- `weight`: When using samples (i.e. the `x` and `y` aesthetics, not `xdist` or `ydist`), optional weights to be applied to each draw.
- `xdist`: When using analytical distributions, distribution to map on the x axis: a **distributional** object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `ydist`: When using analytical distributions, distribution to map on the y axis: a **distributional** object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `dist`: When using analytical distributions, a name of a distribution (e.g. `"norm"`), a **distributional** object (e.g. `dist_normal()`), or a `posterior::rvar()` object. See **Details**.
- `args`: Distribution arguments (`args` or `arg1`, ... `arg9`). See **Details**.

In addition, in their default configuration (paired with `geom_pointinterval()`) the following aesthetics are supported by the underlying geom:

Interval-specific aesthetics

- `xmin`: Left end of the interval sub-geometry (if `orientation = "horizontal"`).
- `xmax`: Right end of the interval sub-geometry (if `orientation = "horizontal"`).
- `ymin`: Lower end of the interval sub-geometry (if `orientation = "vertical"`).
- `ymax`: Upper end of the interval sub-geometry (if `orientation = "vertical"`).

Point-specific aesthetics

- `shape`: Shape type used to draw the **point** sub-geometry.

Color aesthetics

- `colour`: (or `color`) The color of the **interval** and **point** sub-geometries. Use the `slab_color`, `interval_color`, or `point_color` aesthetics (below) to set sub-geometry colors separately.

- `fill`: The fill color of the **slab** and **point** sub-geometries. Use the `slab_fill` or `point_fill` aesthetics (below) to set sub-geometry colors separately.
- `alpha`: The opacity of the **slab**, **interval**, and **point** sub-geometries. Use the `slab_alpha`, `interval_alpha`, or `point_alpha` aesthetics (below) to set sub-geometry colors separately.
- `colour_ramp`: (or `color_ramp`) A secondary scale that modifies the `color` scale to "ramp" to another color. See `scale_colour_ramp()` for examples.
- `fill_ramp`: A secondary scale that modifies the `fill` scale to "ramp" to another color. See `scale_fill_ramp()` for examples.

Line aesthetics

- `linewidth`: Width of the line used to draw the **interval** (except with `geom_slab()`: then it is the width of the **slab**). With composite geometries including an interval and slab, use `slab_linewidth` to set the line width of the **slab** (see below). For **interval**, raw `linewidth` values are transformed according to the `interval_size_domain` and `interval_size_range` parameters of the `geom` (see above).
- `size`: Determines the size of the **point**. If `linewidth` is not provided, `size` will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only `size` and not `linewidth`). Raw `size` values are transformed according to the `interval_size_domain`, `interval_size_range`, and `fatten_point` parameters of the `geom` (see above). Use the `point_size` aesthetic (below) to set sub-geometry size directly without applying the effects of `interval_size_domain`, `interval_size_range`, and `fatten_point`.
- `stroke`: Width of the outline around the **point** sub-geometry.
- `linetype`: Type of line (e.g., `"solid"`, `"dashed"`, etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the `slab_linetype` or `interval_linetype` aesthetics (below) to set sub-geometry line types separately.

Interval-specific color and line override aesthetics

- `interval_colour`: (or `interval_color`) Override for `colour`/`color`: the color of the interval.
- `interval_alpha`: Override for `alpha`: the opacity of the interval.
- `interval_linetype`: Override for `linetype`: the line type of the interval.

Point-specific color and line override aesthetics

- `point_fill`: Override for `fill`: the fill color of the point.
- `point_colour`: (or `point_color`) Override for `colour`/`color`: the outline color of the point.
- `point_alpha`: Override for `alpha`: the opacity of the point.
- `point_size`: Override for `size`: the size of the point.

Deprecated aesthetics

- `interval_size`: Use `interval_linewidth`.

Other aesthetics (these work as in standard `geom`s)

- `width`
- `height`
- `group`

See examples of some of these aesthetics in action in `vignette("slabinterval")`. Learn more about the sub-geom override aesthetics (like `interval_color`) in the scales documentation. Learn more about basic ggplot aesthetics in `vignette("ggplot2-specs")`.

See Also

See `geom_pointinterval()` for the geom underlying this stat. See `stat_slabinterval()` for the stat this shortcut is based on.

Other slabinterval stats: `stat_ccdfinterval()`, `stat_cdfinterval()`, `stat_eye()`, `stat_gradientinterval()`, `stat_halfeye()`, `stat_histinterval()`, `stat_interval()`, `stat_slab()`, `stat_spike()`

Examples

```
library(dplyr)
library(ggplot2)
library(distributional)

theme_set(theme_ggdist())

# ON SAMPLE DATA
set.seed(1234)
df = data.frame(
  group = c("a", "b", "c"),
  value = rnorm(1500, mean = c(5, 7, 9), sd = c(1, 1.5, 1))
)
df %>%
  ggplot(aes(x = value, y = group)) +
  stat_pointinterval()

# ON ANALYTICAL DISTRIBUTIONS
dist_df = data.frame(
  group = c("a", "b", "c"),
  mean =  c(  5,   7,   8),
  sd =    c(  1, 1.5,   1)
)
# Vectorized distribution types, like distributional::dist_normal()
# and posterior::rvar(), can be used with the `xdist` / `ydist` aesthetics
dist_df %>%
  ggplot(aes(y = group, xdist = dist_normal(mean, sd))) +
  stat_pointinterval()
```

stat_ribbon *Multiple-ribbon plot (shortcut stat)*

Description

A combination of `stat_slabinterval()` and `geom_lineribbon()` with sensible defaults for making multiple-ribbon plots. While `geom_lineribbon()` is intended for use on data frames that have already been summarized using a `point_interval()` function, `stat_ribbon()` is intended for use directly on data frames of draws or of analytical distributions, and will perform the summarization using a `point_interval()` function.

Roughly equivalent to:

```
stat_lineribbon(
  show_point = FALSE
)
```

Usage

```
stat_ribbon(
  mapping = NULL,
  data = NULL,
  geom = "lineribbon",
  position = "identity",
  ...,
  .width = c(0.5, 0.8, 0.95),
  point_interval = "median_qi",
  orientation = NA,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping` Set of aesthetic mappings created by `aes()`. 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` <Geom | string> Use to override the default connection between `stat_ribbon()` and `geom_lineribbon()`

`position` <Position | string> Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to `"dodge"` (`position_dodge()`) or `"dodgejust"` (`position_dodgejust()`) can be useful if you have overlapping geometries.

`...` Other arguments passed to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `linewidth = 3` (see **Aesthetics**, below). They may also be parameters to the paired geom/stat. When paired with the default geom, `geom_lineribbon()`, these include:

`step` <scalar logical | string> Should the line/ribbon be drawn as a step function? One of:

- `FALSE` (default): do not draw as a step function.
- `"mid"` (or `TRUE`): draw steps midway between adjacent x values.
- `"hv"`: draw horizontal-then-vertical steps.
- `"vh"`: draw as vertical-then-horizontal steps.

`TRUE` is an alias for `"mid"`, because for a step function with ribbons `"mid"` is reasonable default (for the other two step approaches the ribbons at either the very first or very last x value will not be visible).

`.width` <numeric> The `.width` argument passed to `point_interval`: a vector of probabilities to use that determine the widths of the resulting intervals. If multiple probabilities are provided, multiple intervals per group are generated, each with a different probability interval (and value of the corresponding `.width` and `level` generated variables).

`point_interval` <function | string> A function from the `point_interval()` family (e.g., `median_qi`, `mean_qi`, `mode_hdi`, etc), or a string giving the name of a function from that family (e.g., `"median_qi"`, `"mean_qi"`, `"mode_hdi"`, etc; if a string, the caller's environment is searched for the function, followed by the **ggdist** environment). This function determines the point summary (typically mean, median, or mode) and interval type (quantile interval, `qi`; highest-density interval, `hdi`; or highest-density continuous interval, `hdci`). Output will be converted to the appropriate `x`- or `y`-based aesthetics depending on the value of `orientation`. See the `point_interval()` family of functions for more information.

`orientation` <string> Whether this geom is drawn horizontally or vertically. One of:

- `NA` (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
- `"horizontal"` (or `"y"`): draw horizontally, using the `y` aesthetic to identify different groups. For each group, uses the `x`, `xmin`, `xmax`, and `thickness` aesthetics to draw points, intervals, and slabs.
- `"vertical"` (or `"x"`): draw vertically, using the `x` aesthetic to identify different groups. For each group, uses the `y`, `ymin`, `ymax`, and `thickness` aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for `orientation`, `"x"` can be used as an alias for `"vertical"` and `"y"` as an alias for `"horizontal"` (**ggdist** had an `orientation` parameter before base ggplot did, hence the discrepancy).

na.rm	<scalar logical> 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

To visualize sample data, such as a data distribution, samples from a bootstrap distribution, or a Bayesian posterior, you can supply samples to the x or y aesthetic.

To visualize analytical distributions, you can use the xdist or ydist aesthetic. For historical reasons, you can also use dist to specify the distribution, though this is not recommended as it does not work as well with orientation detection. These aesthetics can be used as follows:

- xdist, ydist, and dist can be any distribution object from the distributional package (dist_normal(), dist_beta(), etc) or can be a posterior::rvar() object. Since these functions are vectorized, other columns can be passed directly to them in an aes() specification; e.g. aes(dist = dist_normal(mu, sigma)) will work if mu and sigma are columns in the input data frame.
- dist can be a character vector giving the distribution name. Then the arg1, ... arg9 aesthetics (or args as a list column) specify distribution arguments. Distribution names should correspond to R functions that have "p", "q", and "d" functions; e.g. "norm" is a valid distribution name because R defines the pnorm(), qnorm(), and dnorm() functions for Normal distributions.

  See the parse_dist() function for a useful way to generate dist and args values from human-readable distribution specs (like "normal(0,1)"). Such specs are also produced by other packages (like the brms::get_prior function in brms); thus, parse_dist() combined with the stats described here can help you visualize the output of those functions.

Value

A ggplot2::Stat representing a multiple-ribbon geometry which can be added to a ggplot() object.

Computed Variables

The following variables are computed by this stat and made available for use in aesthetic specifications (aes()) using the after_stat() function or the after_stat argument of stage():

- x or y: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is x or y depends on orientation
- xmin or ymin: For intervals, the lower end of the interval from the interval function.
- xmax or ymax: For intervals, the upper end of the interval from the interval function.
- .width: For intervals, the interval width as a numeric value in [0, 1]. For slabs, the width of the smallest interval containing that value of the slab.
- level: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.

- pdf: For slabs, the probability density function (PDF). If options("ggdist.experimental.slab_data_in_intervals") is TRUE: For intervals, the PDF at the point summary; intervals also have pdf_min and pdf_max for the PDF at the lower and upper ends of the interval.
- cdf: For slabs, the cumulative distribution function. If options("ggdist.experimental.slab_data_in_intervals") is TRUE: For intervals, the CDF at the point summary; intervals also have cdf_min and cdf_max for the CDF at the lower and upper ends of the interval.

Aesthetics

The line+ribbon stats and geoms have a wide variety of aesthetics that control the appearance of their two sub-geometries: the **line** and the **ribbon**.

These stats support the following aesthetics:

- x: x position of the geometry (when orientation = "vertical"); or sample data to be summarized (when orientation = "horizontal" with sample data).
- y: y position of the geometry (when orientation = "horizontal"); or sample data to be summarized (when orientation = "vertical" with sample data).
- weight: When using samples (i.e. the x and y aesthetics, not xdist or ydist), optional weights to be applied to each draw.
- xdist: When using analytical distributions, distribution to map on the x axis: a **distributional** object (e.g. dist_normal()) or a posterior::rvar() object.
- ydist: When using analytical distributions, distribution to map on the y axis: a **distributional** object (e.g. dist_normal()) or a posterior::rvar() object.
- dist: When using analytical distributions, a name of a distribution (e.g. "norm"), a **distributional** object (e.g. dist_normal()), or a posterior::rvar() object. See **Details**.
- args: Distribution arguments (args or arg1, ... arg9). See **Details**.

In addition, in their default configuration (paired with geom_lineribbon()) the following aesthetics are supported by the underlying geom:

Ribbon-specific aesthetics

- xmin: Left edge of the ribbon sub-geometry (if orientation = "horizontal").
- xmax: Right edge of the ribbon sub-geometry (if orientation = "horizontal").
- ymin: Lower edge of the ribbon sub-geometry (if orientation = "vertical").
- ymax: Upper edge of the ribbon sub-geometry (if orientation = "vertical").
- order: The order in which ribbons are drawn. Ribbons with the smallest mean value of order are drawn first (i.e., will be drawn below ribbons with larger mean values of order). If order is not supplied to geom_lineribbon(), -abs(xmax - xmin) or -abs(ymax - ymax) (depending on orientation) is used, having the effect of drawing the widest (on average) ribbons on the bottom. stat_lineribbon() uses order = after_stat(level) by default, causing the ribbons generated from the largest .width to be drawn on the bottom.

Color aesthetics

- colour: (or color) The color of the **line** sub-geometry.

- fill: The fill color of the **ribbon** sub-geometry.
- alpha: The opacity of the **line** and **ribbon** sub-geometries.
- fill_ramp: A secondary scale that modifies the fill scale to "ramp" to another color. See scale_fill_ramp() for examples.

Other aesthetics (these work as in standard geoms)

- group

See examples of some of these aesthetics in action in vignette("lineribbon"). Learn more about the sub-geom override aesthetics (like interval_color) in the scales documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").

See Also

See geom_lineribbon() for the geom underlying this stat.

Other lineribbon stats: stat_lineribbon()

Examples

```
library(dplyr)
library(ggplot2)
library(distributional)

theme_set(theme_ggdist())

# ON SAMPLE DATA
set.seed(12345)
tibble(
  x = rep(1:10, 100),
  y = rnorm(1000, x)
) %>%
  ggplot(aes(x = x, y = y)) +
  stat_ribbon() +
  scale_fill_brewer()

# ON ANALYTICAL DISTRIBUTIONS
# Vectorized distribution types, like distributional::dist_normal()
# and posterior::rvar(), can be used with the `xdist` / `ydist` aesthetics
tibble(
  x = 1:10,
  sd = seq(1, 3, length.out = 10)
) %>%
  ggplot(aes(x = x, ydist = dist_normal(x, sd))) +
  stat_ribbon() +
  scale_fill_brewer()
```

stat_slab *Slab (ridge) plot (shortcut stat)*

Description

Shortcut version of `stat_slabinterval()` with `geom_slab()` for creating slab (ridge) plots.

Roughly equivalent to:

```
stat_slabinterval(
  aes(size = NULL),
  geom = "slab",
  show_point = FALSE,
  show_interval = FALSE,
  show.legend = NA
)
```

Usage

```
stat_slab(
  mapping = NULL,
  data = NULL,
  geom = "slab",
  position = "identity",
  ...,
  p_limits = c(NA, NA),
  density = "bounded",
  adjust = waiver(),
  trim = waiver(),
  breaks = waiver(),
  align = waiver(),
  outline_bars = waiver(),
  expand = FALSE,
  limits = NULL,
  n = waiver(),
  orientation = NA,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

data
: The data to be displayed in this layer. There are three options:

  If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.

  A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.

  A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).

geom
: <Geom | string> Use to override the default connection between `stat_slab()` and `geom_slab()`

position
: <Position | string> Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to `"dodge"` (`position_dodge()`) or `"dodgejust"` (`position_dodgejust()`) can be useful if you have overlapping geometries.

...
: Other arguments passed to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `linewidth = 3` (see **Aesthetics**, below). They may also be parameters to the paired geom/stat. When paired with the default geom, `geom_slab()`, these include:

  `subscale` <function | string> Sub-scale used to scale values of the `thickness` aesthetic within the groups determined by `normalize`. One of:

  - A function that takes an `x` argument giving a numeric vector of values to be scaled and then returns a thickness vector representing the scaled values, such as `subscale_thickness()` or `subscale_identity()`.
  - A string giving the name of such a function when prefixed with `"subscale_"`; e.g. `"thickness"` or `"identity"`. The value `"thickness"` using the default subscale, which can be modified by setting `subscale_thickness`; see the documentation for that function.

  For a comprehensive discussion and examples of slab scaling and normalization, see the `thickness` scale article.

  `normalize` <string> Groups within which to scale values of the `thickness` aesthetic. One of:

  - `"all"`: normalize so that the maximum height across all data is `1`.
  - `"panels"`: normalize within panels so that the maximum height in each panel is `1`.
  - `"xy"`: normalize within the x/y axis opposite the `orientation` of this geom so that the maximum height at each value of the opposite axis is `1`.
  - `"groups"`: normalize within values of the opposite axis and within each group so that the maximum height in each group is `1`.
  - `"none"`: values are taken as is with no normalization (this should probably only be used with functions whose values are in [0,1], such as CDFs).

  For a comprehensive discussion and examples of slab scaling and normalization, see the `thickness` scale article.

`fill_type` <string> What type of fill to use when the fill color or alpha varies within a slab. One of:

- `"segments"`: breaks up the slab geometry into segments for each unique combination of fill color and alpha value. This approach is supported by all graphics devices and works well for sharp cutoff values, but can give ugly results if a large number of unique fill colors are being used (as in gradients, like in `stat_gradientinterval()`).
- `"gradient"`: a `grid::linearGradient()` is used to create a smooth gradient fill. This works well for large numbers of unique fill colors, but requires R >= 4.1 and is not yet supported on all graphics devices. As of this writing, the `png()` graphics device with `type = "cairo"`, the `svg()` device, the `pdf()` device, and the `ragg::agg_png()` devices are known to support this option. On R < 4.1, this option will fall back to `fill_type = "segments"` with a message.
- `"auto"`: attempts to use `fill_type = "gradient"` if support for it can be auto-detected. On R >= 4.2, support for gradients can be auto-detected on some graphics devices; if support is not detected, this option will fall back to `fill_type = "segments"` (in case of a false negative, `fill_type = "gradient"` can be set explicitly). On R < 4.2, support for gradients cannot be auto-detected, so this will always fall back to `fill_type = "segments"`, in which case you can set `fill_type = "gradient"` explicitly if you are using a graphics device that support gradients.

`subguide` <function | string> Sub-guide used to annotate the `thickness` scale. One of:

- A function that takes a `scale` argument giving a ggplot2::Scale object and an `orientation` argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as `subguide_axis()` (to draw a traditional axis) or `subguide_none()` (to draw no annotation). See `subguide_axis()` for a list of possibilities and examples.
- A string giving the name of such a function when prefixed with `"subguide_"`; e.g. `"axis"` or `"none"`. The values `"slab"`, `"dots"`, and `"spike"` use the default subguide for their geom families (no subguide), which can be modified by setting `subguide_slab`, `subguide_dots`, or `subguide_spike`; see the documentation for those functions.

`p_limits` <length-2 numeric> Probability limits. Used to determine the lower and upper limits of *analytical* distributions (distributions from *samples* ignore this parameter and determine their limits based on the limits of the sample and the value of the `trim` parameter). E.g., if this is `c(.001, .999)`, then a slab is drawn for the distribution from the quantile at `p = .001` to the quantile at `p = .999`. If the lower (respectively upper) limit is `NA`, then the lower (upper) limit will be the minimum (maximum) of the distribution's support if it is finite, and `0.001` (`0.999`) if it is not finite. E.g., if `p_limits` is `c(NA, NA)`, on a gamma distribution the effective value of `p_limits` would be `c(0, .999)` since the gamma distribution is defined on `(0, Inf)`; whereas on a normal distribution it would be equivalent to `c(.001, .999)` since the normal distribution is defined on `(-Inf, Inf)`.

density
: <function | string> Density estimator for sample data. One of:

- A function which takes a numeric vector and returns a list with elements x (giving grid points for the density estimator) and y (the corresponding densities). **ggdist** provides a family of functions following this format, including density_unbounded() and density_bounded(). This format is also compatible with stats::density().
- A string giving the suffix of a function name that starts with "density_"; e.g. "bounded" for [density_bounded()], "unbounded" for [density_unbounded()], or "histogram" for density_histogram(). Defaults to "bounded", i.e. density_bounded(), which estimates the bounds from the data and then uses a bounded density estimator based on the reflection method.

adjust
: <scalar numeric | waiver> Passed to density (e.g. density_bounded()): Value to multiply the bandwidth of the density estimator by. Default waiver() defers to the default of the density estimator, which is usually 1.

trim
: <scalar logical | waiver> Passed to density (e.g. density_bounded()): Should the density estimate be trimmed to the range of the data? Default waiver() defers to the default of the density estimator, which is usually TRUE.

breaks
: <numeric | function | string | waiver> Passed to density (e.g. density_histogram()): Determines the breakpoints defining bins. Default waiver() defers to the default of the density estimator, which is usually "Scott". Similar to (but not exactly the same as) the breaks argument to graphics::hist(). One of:

- A scalar (length-1) numeric giving the number of bins
- A vector numeric giving the breakpoints between histogram bins
- A function taking x and weights and returning either the number of bins or a vector of breakpoints
- A string giving the suffix of a function that starts with "breaks_". **ggdist** provides weighted implementations of the "Sturges", "Scott", and "FD" break-finding algorithms from graphics::hist(), as well as breaks_fixed() for manually setting the bin width. See breaks.

For example, breaks = "Sturges" will use the breaks_Sturges() algorithm, breaks = 9 will create 9 bins, and breaks = breaks_fixed(width = 1) will set the bin width to 1.

align
: <scalar numeric | function | string | waiver> Passed to density (e.g. density_histogram()): Determines how to align the breakpoints defining bins. Default waiver() defers to the default of the density estimator, which is usually "none" (performs no alignment). One of:

- A scalar (length-1) numeric giving an offset that is subtracted from the breaks. The offset must be between 0 and the bin width.
- A function taking a sorted vector of breaks (bin edges) and returning an offset to subtract from the breaks.
- A string giving the suffix of a function that starts with "align_" used to determine the alignment, such as align_none(), align_boundary(), or align_center().

For example, align = "none" will provide no alignment, align = align_center(at = 0) will center a bin on 0, and align = align_boundary(at = 0) will align a bin edge on 0.

`outline_bars` <scalar logical | waiver> Passed to density (e.g. `density_histogram()`) and also used for discrete analytical distributions (whose slabs are drawn as histograms). Determines if outlines in between the bars are drawn. If `waiver()` or `FALSE` (the default), the outline is drawn only along the tops of the bars. If `TRUE`, outlines in between bars are also drawn (though you may have to set the `slab_color` or `color` aesthetic to see the outlines).

`expand` <logical> For sample data, should the slab be expanded to the limits of the scale? Default `FALSE`. Can be a length-two logical vector to control expansion to the lower and upper limit respectively.

`limits` <length-2 numeric> 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` <scalar numeric> Number of points at which to evaluate the function that defines the slab. Also passed to `density` (e.g. `density_bounded()`). Default `waiver()` uses the value `501` for analytical distributions and defers to the default of the density estimator for sample-based distributions, which is also usually `501`.

`orientation` <string> Whether this geom is drawn horizontally or vertically. One of:

- `NA` (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
- `"horizontal"` (or `"y"`): draw horizontally, using the `y` aesthetic to identify different groups. For each group, uses the `x`, `xmin`, `xmax`, and `thickness` aesthetics to draw points, intervals, and slabs.
- `"vertical"` (or `"x"`): draw vertically, using the `x` aesthetic to identify different groups. For each group, uses the `y`, `ymin`, `ymax`, and `thickness` aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for `orientation`, `"x"` can be used as an alias for `"vertical"` and `"y"` as an alias for `"horizontal"` (**ggdist** had an `orientation` parameter before base ggplot did, hence the discrepancy).

`na.rm` <scalar logical> 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? 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). 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

To visualize sample data, such as a data distribution, samples from a bootstrap distribution, or a Bayesian posterior, you can supply samples to the x or y aesthetic.

To visualize analytical distributions, you can use the xdist or ydist aesthetic. For historical reasons, you can also use dist to specify the distribution, though this is not recommended as it does not work as well with orientation detection. These aesthetics can be used as follows:

- xdist, ydist, and dist can be any distribution object from the distributional package (dist_normal(), dist_beta(), etc) or can be a posterior::rvar() object. Since these functions are vectorized, other columns can be passed directly to them in an aes() specification; e.g. aes(dist = dist_normal(mu, sigma)) will work if mu and sigma are columns in the input data frame.
- dist can be a character vector giving the distribution name. Then the arg1, ... arg9 aesthetics (or args as a list column) specify distribution arguments. Distribution names should correspond to R functions that have "p", "q", and "d" functions; e.g. "norm" is a valid distribution name because R defines the pnorm(), qnorm(), and dnorm() functions for Normal distributions.

  See the parse_dist() function for a useful way to generate dist and args values from human-readable distribution specs (like "normal(0,1)"). Such specs are also produced by other packages (like the brms::get_prior function in brms); thus, parse_dist() combined with the stats described here can help you visualize the output of those functions.

Value

A ggplot2::Stat representing a slab (ridge) geometry which can be added to a ggplot() object.

Computed Variables

The following variables are computed by this stat and made available for use in aesthetic specifications (aes()) using the after_stat() function or the after_stat argument of stage():

- x or y: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is x or y depends on orientation
- xmin or ymin: For intervals, the lower end of the interval from the interval function.
- xmax or ymax: For intervals, the upper end of the interval from the interval function.
- .width: For intervals, the interval width as a numeric value in [0, 1]. For slabs, the width of the smallest interval containing that value of the slab.
- level: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.
- pdf: For slabs, the probability density function (PDF). If options("ggdist.experimental.slab_data_in_interval is TRUE: For intervals, the PDF at the point summary; intervals also have pdf_min and pdf_max for the PDF at the lower and upper ends of the interval.
- cdf: For slabs, the cumulative distribution function. If options("ggdist.experimental.slab_data_in_intervals' is TRUE: For intervals, the CDF at the point summary; intervals also have cdf_min and cdf_max for the CDF at the lower and upper ends of the interval.
- n: For slabs, the number of data points summarized into that slab. If the slab was created from an analytical distribution via the xdist, ydist, or dist aesthetic, n will be Inf.

- f: (deprecated) For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by slab_type. Instead of using slab_type to change f and then mapping f onto an aesthetic, it is now recommended to simply map the corresponding computed variable (e.g. pdf, cdf, or 1 - cdf) directly onto the desired aesthetic.

Aesthetics

The slab+interval stats and geoms have a wide variety of aesthetics that control the appearance of their three sub-geometries: the **slab**, the **point**, and the **interval**.

These stats support the following aesthetics:

- x: x position of the geometry (when orientation = "vertical"); or sample data to be summarized (when orientation = "horizontal" with sample data).
- y: y position of the geometry (when orientation = "horizontal"); or sample data to be summarized (when orientation = "vertical" with sample data).
- weight: When using samples (i.e. the x and y aesthetics, not xdist or ydist), optional weights to be applied to each draw.
- xdist: When using analytical distributions, distribution to map on the x axis: a **distributional** object (e.g. dist_normal()) or a posterior::rvar() object.
- ydist: When using analytical distributions, distribution to map on the y axis: a **distributional** object (e.g. dist_normal()) or a posterior::rvar() object.
- dist: When using analytical distributions, a name of a distribution (e.g. "norm"), a **distributional** object (e.g. dist_normal()), or a posterior::rvar() object. See **Details**.
- args: Distribution arguments (args or arg1, ... arg9). See **Details**.

In addition, in their default configuration (paired with geom_slab()) the following aesthetics are supported by the underlying geom:

Slab-specific aesthetics

- thickness: The thickness of the slab at each x value (if orientation = "horizontal") or y value (if orientation = "vertical") of the slab.
- side: Which side to place the slab on. "topright", "top", and "right" are synonyms which cause the slab to be drawn on the top or the right depending on if orientation is "horizontal" or "vertical". "bottomleft", "bottom", and "left" are synonyms which cause the slab to be drawn on the bottom or the left depending on if orientation is "horizontal" or "vertical". "topleft" causes the slab to be drawn on the top or the left, and "bottomright" causes the slab to be drawn on the bottom or the right. "both" draws the slab mirrored on both sides (as in a violin plot).
- scale: What proportion of the region allocated to this geom to use to draw the slab. If scale = 1, slabs that use the maximum range will just touch each other. Default is 0.9 to leave some space between adjacent slabs. For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.
- 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.

Color aesthetics

- colour: (or color) The color of the **interval** and **point** sub-geometries. Use the slab_color, interval_color, or point_color aesthetics (below) to set sub-geometry colors separately.
- fill: The fill color of the **slab** and **point** sub-geometries. Use the slab_fill or point_fill aesthetics (below) to set sub-geometry colors separately.
- alpha: The opacity of the **slab**, **interval**, and **point** sub-geometries. Use the slab_alpha, interval_alpha, or point_alpha aesthetics (below) to set sub-geometry colors separately.
- colour_ramp: (or color_ramp) A secondary scale that modifies the color scale to "ramp" to another color. See scale_colour_ramp() for examples.
- fill_ramp: A secondary scale that modifies the fill scale to "ramp" to another color. See scale_fill_ramp() for examples.

Line aesthetics

- linewidth: Width of the line used to draw the **interval** (except with geom_slab(): then it is the width of the **slab**). With composite geometries including an interval and slab, use slab_linewidth to set the line width of the **slab** (see below). For **interval**, raw linewidth values are transformed according to the interval_size_domain and interval_size_range parameters of the geom (see above).
- size: Determines the size of the **point**. If linewidth is not provided, size will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only size and not linewidth). Raw size values are transformed according to the interval_size_domain, interval_size_range, and fatten_point parameters of the geom (see above). Use the point_size aesthetic (below) to set sub-geometry size directly without applying the effects of interval_size_domain, interval_size_range, and fatten_point.
- stroke: Width of the outline around the **point** sub-geometry.
- linetype: Type of line (e.g., "solid", "dashed", etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the slab_linetype or interval_linetype aesthetics (below) to set sub-geometry line types separately.

Slab-specific color and line override aesthetics

- slab_fill: Override for fill: the fill color of the slab.
- slab_colour: (or slab_color) Override for colour/color: the outline color of the slab.
- slab_alpha: Override for alpha: the opacity of the slab.
- slab_linewidth: Override for linwidth: the width of the outline of the slab.
- slab_linetype: Override for linetype: the line type of the outline of the slab.

Deprecated aesthetics

- slab_size: Use slab_linewidth.

Other aesthetics (these work as in standard geoms)

- width

- height
- group

See examples of some of these aesthetics in action in vignette("slabinterval"). Learn more about the sub-geom override aesthetics (like interval_color) in the scales documentation. Learn more about basic ggplot aesthetics in vignette("ggplot2-specs").

See Also

See geom_slab() for the geom underlying this stat. See stat_slabinterval() for the stat this shortcut is based on.

Other slabinterval stats: stat_ccdfinterval(), stat_cdfinterval(), stat_eye(), stat_gradientinterval(), stat_halfeye(), stat_histinterval(), stat_interval(), stat_pointinterval(), stat_spike()

Examples

```
library(dplyr)
library(ggplot2)
library(distributional)

theme_set(theme_ggdist())

# ON SAMPLE DATA
set.seed(1234)
df = data.frame(
  group = c("a", "b", "c"),
  value = rnorm(1500, mean = c(5, 7, 9), sd = c(1, 1.5, 1))
)
df %>%
  ggplot(aes(x = value, y = group)) +
  stat_slab()

# ON ANALYTICAL DISTRIBUTIONS
dist_df = data.frame(
  group = c("a", "b", "c"),
  mean =  c(  5,   7,   8),
  sd =    c(  1, 1.5,   1)
)
# Vectorized distribution types, like distributional::dist_normal()
# and posterior::rvar(), can be used with the `xdist` / `ydist` aesthetics
dist_df %>%
  ggplot(aes(y = group, xdist = dist_normal(mean, sd))) +
  stat_slab()

# RIDGE PLOTS
# "ridge" plots can be created by expanding the slabs to the limits of the plot
# (expand = TRUE), allowing the density estimator to be nonzero outside the
# limits of the data (trim = FALSE), and increasing the height of the slabs.
data.frame(
  group = letters[1:3],
  value = rnorm(3000, 3:1)
) %>%
```

```
ggplot(aes(y = group, x = value)) +
stat_slab(color = "black", expand = TRUE, trim = FALSE, height = 2)
```

stat_slabinterval *Slab + interval plots for sample data and analytical distributions (ggplot stat)*

Description

"Meta" stat for computing distribution functions (densities or CDFs) + intervals for use with geom_slabinterval(). Useful for creating eye plots, half-eye plots, CCDF bar plots, gradient plots, histograms, and more. Sample data can be supplied to the x and y aesthetics or analytical distributions (in a variety of formats) can be supplied to the xdist and ydist aesthetics. See **Details**.

Usage

```
stat_slabinterval(
  mapping = NULL,
  data = NULL,
  geom = "slabinterval",
  position = "identity",
  ...,
  p_limits = c(NA, NA),
  density = "bounded",
  adjust = waiver(),
  trim = waiver(),
  breaks = waiver(),
  align = waiver(),
  outline_bars = waiver(),
  expand = FALSE,
  point_interval = "median_qi",
  limits = NULL,
  n = waiver(),
  .width = c(0.66, 0.95),
  orientation = NA,
  na.rm = FALSE,
  show.legend = c(size = FALSE),
  inherit.aes = TRUE
)
```

Arguments

mapping Set of aesthetic mappings created by aes(). 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` <Geom | string> Use to override the default connection between `stat_slabinterval()` and `geom_slabinterval()`.

`position` <Position | string> Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to `"dodge"` (`position_dodge()`) or `"dodgejust"` (`position_dodgejust()`) can be useful if you have overlapping geometries.

`...` Other arguments passed to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `linewidth = 3` (see **Aesthetics**, below). They may also be parameters to the paired geom/stat. When paired with the default geom, `geom_slabinterval()`, these include:

`subscale` <function | string> Sub-scale used to scale values of the `thickness` aesthetic within the groups determined by `normalize`. One of:

- A function that takes an `x` argument giving a numeric vector of values to be scaled and then returns a thickness vector representing the scaled values, such as `subscale_thickness()` or `subscale_identity()`.
- A string giving the name of such a function when prefixed with `"subscale_"`; e.g. `"thickness"` or `"identity"`. The value `"thickness"` using the default subscale, which can be modified by setting `subscale_thickness`; see the documentation for that function.

For a comprehensive discussion and examples of slab scaling and normalization, see the `thickness` scale article.

`normalize` <string> Groups within which to scale values of the `thickness` aesthetic. One of:

- `"all"`: normalize so that the maximum height across all data is `1`.
- `"panels"`: normalize within panels so that the maximum height in each panel is `1`.
- `"xy"`: normalize within the x/y axis opposite the `orientation` of this geom so that the maximum height at each value of the opposite axis is `1`.
- `"groups"`: normalize within values of the opposite axis and within each group so that the maximum height in each group is `1`.
- `"none"`: values are taken as is with no normalization (this should probably only be used with functions whose values are in [0,1], such as CDFs).

For a comprehensive discussion and examples of slab scaling and normalization, see the `thickness` scale article.

fill_type <string> What type of fill to use when the fill color or alpha varies within a slab. One of:

- "segments": breaks up the slab geometry into segments for each unique combination of fill color and alpha value. This approach is supported by all graphics devices and works well for sharp cutoff values, but can give ugly results if a large number of unique fill colors are being used (as in gradients, like in stat_gradientinterval()).
- "gradient": a grid::linearGradient() is used to create a smooth gradient fill. This works well for large numbers of unique fill colors, but requires R >= 4.1 and is not yet supported on all graphics devices. As of this writing, the png() graphics device with type = "cairo", the svg() device, the pdf() device, and the ragg::agg_png() devices are known to support this option. On R < 4.1, this option will fall back to fill_type = "segments" with a message.
- "auto": attempts to use fill_type = "gradient" if support for it can be auto-detected. On R >= 4.2, support for gradients can be auto-detected on some graphics devices; if support is not detected, this option will fall back to fill_type = "segments" (in case of a false negative, fill_type = "gradient" can be set explicitly). On R < 4.2, support for gradients cannot be auto-detected, so this will always fall back to fill_type = "segments", in which case you can set fill_type = "gradient" explicitly if you are using a graphics device that support gradients.

interval_size_domain <length-2 numeric> Minimum and maximum of the values of the size and linewidth aesthetics that will be translated into actual sizes for intervals drawn according to interval_size_range (see the documentation for that argument.)

interval_size_range <length-2 numeric> This geom scales the raw size aesthetic values when drawing interval and point sizes, as they tend to be too thick when using the default settings of scale_size_continuous(), which give sizes with a range of c(1, 6). The interval_size_domain value indicates the input domain of raw size values (typically this should be equal to the value of the range argument of the scale_size_continuous() function), and interval_size_range indicates the desired output range of the size values (the min and max of the actual sizes used to draw intervals). Most of the time it is not recommended to change the value of this argument, as it may result in strange scaling of legends; this argument is a holdover from earlier versions that did not have size aesthetics targeting the point and interval separately. If you want to adjust the size of the interval or points separately, you can also use the linewidth or point_size aesthetics; see sub-geometry-scales.

fatten_point <scalar numeric> 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.

arrow <arrow | NULL> Type of arrow heads to use on the interval, or NULL for no arrows.

subguide <function | string> Sub-guide used to annotate the thickness scale. One of:

- A function that takes a scale argument giving a ggplot2::Scale object and an orientation argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as subguide_axis() (to draw a traditional axis) or subguide_none() (to draw no annotation). See subguide_axis() for a list of possibilities and examples.
- A string giving the name of such a function when prefixed with "subguide_"; e.g. "axis" or "none". The values "slab", "dots", and "spike" use the default subguide for their geom families (no subguide), which can be modified by setting subguide_slab, subguide_dots, or subguide_spike; see the documentation for those functions.

p_limits <length-2 numeric> Probability limits. Used to determine the lower and upper limits of *analytical* distributions (distributions from *samples* ignore this parameter and determine their limits based on the limits of the sample and the value of the trim parameter). E.g., if this is c(.001, .999), then a slab is drawn for the distribution from the quantile at p = .001 to the quantile at p = .999. If the lower (respectively upper) limit is NA, then the lower (upper) limit will be the minimum (maximum) of the distribution's support if it is finite, and 0.001 (0.999) if it is not finite. E.g., if p_limits is c(NA, NA), on a gamma distribution the effective value of p_limits would be c(0, .999) since the gamma distribution is defined on (0, Inf); whereas on a normal distribution it would be equivalent to c(.001, .999) since the normal distribution is defined on (-Inf, Inf).

density <function | string> Density estimator for sample data. One of:

- A function which takes a numeric vector and returns a list with elements x (giving grid points for the density estimator) and y (the corresponding densities). **ggdist** provides a family of functions following this format, including density_unbounded() and density_bounded(). This format is also compatible with stats::density().
- A string giving the suffix of a function name that starts with "density_"; e.g. "bounded" for [density_bounded()], "unbounded" for [density_unbounded()], or "histogram" for density_histogram(). Defaults to "bounded", i.e. density_bounded(), which estimates the bounds from the data and then uses a bounded density estimator based on the reflection method.

adjust <scalar numeric | waiver> Passed to density (e.g. density_bounded()): Value to multiply the bandwidth of the density estimator by. Default waiver() defers to the default of the density estimator, which is usually 1.

trim <scalar logical | waiver> Passed to density (e.g. density_bounded()): Should the density estimate be trimmed to the range of the data? Default waiver() defers to the default of the density estimator, which is usually TRUE.

breaks <numeric | function | string | waiver> Passed to density (e.g. density_histogram()): Determines the breakpoints defining bins. Default waiver() defers to the default of the density estimator, which is usually "Scott". Similar to (but not exactly the same as) the breaks argument to graphics::hist(). One of:

- A scalar (length-1) numeric giving the number of bins

- A vector numeric giving the breakpoints between histogram bins
- A function taking x and weights and returning either the number of bins or a vector of breakpoints
- A string giving the suffix of a function that starts with "breaks_". **ggdist** provides weighted implementations of the "Sturges", "Scott", and "FD" break-finding algorithms from graphics::hist(), as well as breaks_fixed() for manually setting the bin width. See breaks.

For example, breaks = "Sturges" will use the breaks_Sturges() algorithm, breaks = 9 will create 9 bins, and breaks = breaks_fixed(width = 1) will set the bin width to 1.

align
<scalar numeric | function | string | waiver> Passed to density (e.g. density_histogram()): Determines how to align the breakpoints defining bins. Default waiver() defers to the default of the density estimator, which is usually "none" (performs no alignment). One of:

- A scalar (length-1) numeric giving an offset that is subtracted from the breaks. The offset must be between 0 and the bin width.
- A function taking a sorted vector of breaks (bin edges) and returning an offset to subtract from the breaks.
- A string giving the suffix of a function that starts with "align_" used to determine the alignment, such as align_none(), align_boundary(), or align_center().

For example, align = "none" will provide no alignment, align = align_center(at = 0) will center a bin on 0, and align = align_boundary(at = 0) will align a bin edge on 0.

outline_bars
<scalar logical | waiver> Passed to density (e.g. density_histogram()) and also used for discrete analytical distributions (whose slabs are drawn as histograms). Determines if outlines in between the bars are drawn. If waiver() or FALSE (the default), the outline is drawn only along the tops of the bars. If TRUE, outlines in between bars are also drawn (though you may have to set the slab_color or color aesthetic to see the outlines).

expand
<logical> For sample data, should the slab be expanded to the limits of the scale? Default FALSE. Can be a length-two logical vector to control expansion to the lower and upper limit respectively.

point_interval
<function | string> A function from the point_interval() family (e.g., median_qi, mean_qi, mode_hdi, etc), or a string giving the name of a function from that family (e.g., "median_qi", "mean_qi", "mode_hdi", etc; if a string, the caller's environment is searched for the function, followed by the **ggdist** environment). This function determines the point summary (typically mean, median, or mode) and interval type (quantile interval, qi; highest-density interval, hdi; or highest-density continuous interval, hdci). Output will be converted to the appropriate x- or y-based aesthetics depending on the value of orientation. See the point_interval() family of functions for more information.

limits
<length-2 numeric> 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` <scalar numeric> Number of points at which to evaluate the function that defines the slab. Also passed to `density` (e.g. `density_bounded()`). Default `waiver()` uses the value `501` for analytical distributions and defers to the default of the density estimator for sample-based distributions, which is also usually `501`.

`.width` <numeric> The `.width` argument passed to `point_interval`: a vector of probabilities to use that determine the widths of the resulting intervals. If multiple probabilities are provided, multiple intervals per group are generated, each with a different probability interval (and value of the corresponding `.width` and `level` generated variables).

`orientation` <string> Whether this geom is drawn horizontally or vertically. One of:

- `NA` (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
- `"horizontal"` (or `"y"`): draw horizontally, using the `y` aesthetic to identify different groups. For each group, uses the `x`, `xmin`, `xmax`, and `thickness` aesthetics to draw points, intervals, and slabs.
- `"vertical"` (or `"x"`): draw vertically, using the `x` aesthetic to identify different groups. For each group, uses the `y`, `ymin`, `ymax`, and `thickness` aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for `orientation`, `"x"` can be used as an alias for `"vertical"` and `"y"` as an alias for `"horizontal"` (**ggdist** had an `orientation` parameter before base ggplot did, hence the discrepancy).

`na.rm` <scalar logical> 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? 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). 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

A highly configurable stat for generating a variety of plots that combine a "slab" that describes a distribution plus a point summary and any number of intervals. Several "shortcut" stats are provided which combine multiple options to create useful geoms, particularly *eye plots* (a violin plot of density plus interval), *half-eye plots* (a density plot plus interval), *CCDF bar plots* (a complementary CDF plus interval), and *gradient plots* (a density encoded in color alpha plus interval).

The shortcut stats include:

- `stat_eye()`: Eye plots (violin + interval)
- `stat_halfeye()`: Half-eye plots (density + interval)
- `stat_ccdfinterval()`: CCDF bar plots (CCDF + interval)
- `stat_cdfinterval()`: CDF bar plots (CDF + interval)
- `stat_gradientinterval()`: Density gradient + interval plots
- `stat_slab()`: Density plots
- `stat_histinterval()`: Histogram + interval plots
- `stat_pointinterval()`: Point + interval plots
- `stat_interval()`: Interval plots

To visualize sample data, such as a data distribution, samples from a bootstrap distribution, or a Bayesian posterior, you can supply samples to the `x` or `y` aesthetic.

To visualize analytical distributions, you can use the `xdist` or `ydist` aesthetic. For historical reasons, you can also use `dist` to specify the distribution, though this is not recommended as it does not work as well with orientation detection. These aesthetics can be used as follows:

- `xdist`, `ydist`, and `dist` can be any distribution object from the distributional package (`dist_normal()`, `dist_beta()`, etc) or can be a `posterior::rvar()` object. Since these functions are vectorized, other columns can be passed directly to them in an `aes()` specification; e.g. `aes(dist = dist_normal(mu, sigma))` will work if `mu` and `sigma` are columns in the input data frame.
- `dist` can be a character vector giving the distribution name. Then the `arg1`, ... `arg9` aesthetics (or `args` as a list column) specify distribution arguments. Distribution names should correspond to R functions that have `"p"`, `"q"`, and `"d"` functions; e.g. `"norm"` is a valid distribution name because R defines the `pnorm()`, `qnorm()`, and `dnorm()` functions for Normal distributions.

  See the `parse_dist()` function for a useful way to generate `dist` and `args` values from human-readable distribution specs (like `"normal(0,1)"`). Such specs are also produced by other packages (like the `brms::get_prior` function in brms); thus, `parse_dist()` combined with the stats described here can help you visualize the output of those functions.

Value

A ggplot2::Stat representing a slab or combined slab+interval geometry which can be added to a `ggplot()` object.

Computed Variables

The following variables are computed by this stat and made available for use in aesthetic specifications (`aes()`) using the `after_stat()` function or the `after_stat` argument of `stage()`:

- `x` or `y`: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is `x` or `y` depends on `orientation`
- `xmin` or `ymin`: For intervals, the lower end of the interval from the interval function.
- `xmax` or `ymax`: For intervals, the upper end of the interval from the interval function.
- `.width`: For intervals, the interval width as a numeric value in `[0, 1]`. For slabs, the width of the smallest interval containing that value of the slab.

- `level`: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.
- `pdf`: For slabs, the probability density function (PDF). If `options("ggdist.experimental.slab_data_in_interval` is `TRUE`: For intervals, the PDF at the point summary; intervals also have `pdf_min` and `pdf_max` for the PDF at the lower and upper ends of the interval.
- `cdf`: For slabs, the cumulative distribution function. If `options("ggdist.experimental.slab_data_in_intervals'` is `TRUE`: For intervals, the CDF at the point summary; intervals also have `cdf_min` and `cdf_max` for the CDF at the lower and upper ends of the interval.
- `n`: For slabs, the number of data points summarized into that slab. If the slab was created from an analytical distribution via the `xdist`, `ydist`, or `dist` aesthetic, `n` will be `Inf`.
- `f`: (deprecated) For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by `slab_type`. Instead of using `slab_type` to change `f` and then mapping `f` onto an aesthetic, it is now recommended to simply map the corresponding computed variable (e.g. `pdf`, `cdf`, or `1 - cdf`) directly onto the desired aesthetic.

Aesthetics

The slab+interval `stats` and `geoms` have a wide variety of aesthetics that control the appearance of their three sub-geometries: the **slab**, the **point**, and the **interval**.

These `stats` support the following aesthetics:

- `x`: x position of the geometry (when orientation = `"vertical"`); or sample data to be summarized (when `orientation = "horizontal"` with sample data).
- `y`: y position of the geometry (when orientation = `"horizontal"`); or sample data to be summarized (when `orientation = "vertical"` with sample data).
- `weight`: When using samples (i.e. the `x` and `y` aesthetics, not `xdist` or `ydist`), optional weights to be applied to each draw.
- `xdist`: When using analytical distributions, distribution to map on the x axis: a **distributional** object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `ydist`: When using analytical distributions, distribution to map on the y axis: a **distributional** object (e.g. `dist_normal()`) or a `posterior::rvar()` object.
- `dist`: When using analytical distributions, a name of a distribution (e.g. `"norm"`), a **distributional** object (e.g. `dist_normal()`), or a `posterior::rvar()` object. See **Details**.
- `args`: Distribution arguments (`args` or `arg1`, ... `arg9`). See **Details**.

In addition, in their default configuration (paired with `geom_slabinterval()`) the following aesthetics are supported by the underlying geom:

Slab-specific aesthetics

- `thickness`: The thickness of the slab at each `x` value (if `orientation = "horizontal"`) or `y` value (if `orientation = "vertical"`) of the slab.
- `side`: Which side to place the slab on. `"topright"`, `"top"`, and `"right"` are synonyms which cause the slab to be drawn on the top or the right depending on if `orientation` is `"horizontal"` or `"vertical"`. `"bottomleft"`, `"bottom"`, and `"left"` are synonyms which cause the slab to be drawn on the bottom or the left depending on if `orientation`

is "horizontal" or "vertical". "topleft" causes the slab to be drawn on the top or the left, and "bottomright" causes the slab to be drawn on the bottom or the right. "both" draws the slab mirrored on both sides (as in a violin plot).

- scale: What proportion of the region allocated to this geom to use to draw the slab. If scale = 1, slabs that use the maximum range will just touch each other. Default is 0.9 to leave some space between adjacent slabs. For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.
- justification: Justification of the interval relative to the slab, where 0 indicates bottom/left justification and 1 indicates top/right justification (depending on orientation). If justification is NULL (the default), then it is set automatically based on the value of side: when side is "top"/"right" justification is set to 0, when side is "bottom"/"left" justification is set to 1, and when side is "both" justification is set to 0.5.
- datatype: When using composite geoms directly without a stat (e.g. geom_slabinterval()), datatype is used to indicate which part of the geom a row in the data targets: rows with datatype = "slab" target the slab portion of the geometry and rows with datatype = "interval" target the interval portion of the geometry. This is set automatically when using ggdist stats.

Interval-specific aesthetics

- xmin: Left end of the interval sub-geometry (if orientation = "horizontal").
- xmax: Right end of the interval sub-geometry (if orientation = "horizontal").
- ymin: Lower end of the interval sub-geometry (if orientation = "vertical").
- ymax: Upper end of the interval sub-geometry (if orientation = "vertical").

Point-specific aesthetics

- shape: Shape type used to draw the **point** sub-geometry.

Color aesthetics

- colour: (or color) The color of the **interval** and **point** sub-geometries. Use the slab_color, interval_color, or point_color aesthetics (below) to set sub-geometry colors separately.
- fill: The fill color of the **slab** and **point** sub-geometries. Use the slab_fill or point_fill aesthetics (below) to set sub-geometry colors separately.
- alpha: The opacity of the **slab**, **interval**, and **point** sub-geometries. Use the slab_alpha, interval_alpha, or point_alpha aesthetics (below) to set sub-geometry colors separately.
- colour_ramp: (or color_ramp) A secondary scale that modifies the color scale to "ramp" to another color. See scale_colour_ramp() for examples.
- fill_ramp: A secondary scale that modifies the fill scale to "ramp" to another color. See scale_fill_ramp() for examples.

Line aesthetics

- linewidth: Width of the line used to draw the **interval** (except with geom_slab(): then it is the width of the **slab**). With composite geometries including an interval and slab, use slab_linewidth to set the line width of the **slab** (see below). For **interval**, raw linewidth values are transformed according to the interval_size_domain and interval_size_range parameters of the geom (see above).

- `size`: Determines the size of the **point**. If `linewidth` is not provided, `size` will also determines the width of the line used to draw the **interval** (this allows line width and point size to be modified together by setting only `size` and not `linewidth`). Raw `size` values are transformed according to the `interval_size_domain`, `interval_size_range`, and `fatten_point` parameters of the `geom` (see above). Use the `point_size` aesthetic (below) to set sub-geometry size directly without applying the effects of `interval_size_domain`, `interval_size_range`, and `fatten_point`.
- `stroke`: Width of the outline around the **point** sub-geometry.
- `linetype`: Type of line (e.g., `"solid"`, `"dashed"`, etc) used to draw the **interval** and the outline of the **slab** (if it is visible). Use the `slab_linetype` or `interval_linetype` aesthetics (below) to set sub-geometry line types separately.

Slab-specific color and line override aesthetics

- `slab_fill`: Override for `fill`: the fill color of the slab.
- `slab_colour`: (or `slab_color`) Override for `colour`/`color`: the outline color of the slab.
- `slab_alpha`: Override for `alpha`: the opacity of the slab.
- `slab_linewidth`: Override for `linwidth`: the width of the outline of the slab.
- `slab_linetype`: Override for `linetype`: the line type of the outline of the slab.

Interval-specific color and line override aesthetics

- `interval_colour`: (or `interval_color`) Override for `colour`/`color`: the color of the interval.
- `interval_alpha`: Override for `alpha`: the opacity of the interval.
- `interval_linetype`: Override for `linetype`: the line type of the interval.

Point-specific color and line override aesthetics

- `point_fill`: Override for `fill`: the fill color of the point.
- `point_colour`: (or `point_color`) Override for `colour`/`color`: the outline color of the point.
- `point_alpha`: Override for `alpha`: the opacity of the point.
- `point_size`: Override for `size`: the size of the point.

Deprecated aesthetics

- `slab_size`: Use `slab_linewidth`.
- `interval_size`: Use `interval_linewidth`.

Other aesthetics (these work as in standard `geom`s)

- `width`
- `height`
- `group`

See examples of some of these aesthetics in action in `vignette("slabinterval")`. Learn more about the sub-geom override aesthetics (like `interval_color`) in the scales documentation. Learn more about basic ggplot aesthetics in `vignette("ggplot2-specs")`.

See Also

See `geom_slabinterval()` for more information on the geom these stats use by default and some of the options it has. See `vignette("slabinterval")` for a variety of examples of use.

Examples

```
library(dplyr)
library(ggplot2)
library(distributional)

theme_set(theme_ggdist())


# EXAMPLES ON SAMPLE DATA
set.seed(1234)
df = data.frame(
  group = c("a", "b", "c", "c", "c"),
  value = rnorm(2500, mean = c(5, 7, 9, 9, 9), sd = c(1, 1.5, 1, 1, 1))
)

# here are vertical eyes:
df %>%
  ggplot(aes(x = group, y = value)) +
  stat_eye()

# note the sample size is not automatically incorporated into the
# area of the densities in case one wishes to plot densities against
# a reference (e.g. a prior distribution).
# But you may wish to account for sample size if using these geoms
# for something other than visualizing posteriors; in which case
# you can use after_stat(f*n):
df %>%
  ggplot(aes(x = group, y = value)) +
  stat_eye(aes(thickness = after_stat(pdf*n)))


# EXAMPLES ON ANALYTICAL DISTRIBUTIONS

dist_df = tribble(
  ~group, ~subgroup, ~mean, ~sd,
  "a",          "h",     5,   1,
  "b",          "h",     7,   1.5,
  "c",          "h",     8,   1,
  "c",          "i",     9,   1,
  "c",          "j",     7,   1
)

# Using functions from the distributional package (like dist_normal()) with the
# dist aesthetic can lead to more compact/expressive specifications

dist_df %>%
  ggplot(aes(x = group, ydist = dist_normal(mean, sd), fill = subgroup)) +
```

```
  stat_eye(position = "dodge")

# using the old character vector + args approach
dist_df %>%
  ggplot(aes(x = group, dist = "norm", arg1 = mean, arg2 = sd, fill = subgroup)) +
  stat_eye(position = "dodge")

# the stat_slabinterval family applies a Jacobian adjustment to densities
# when plotting on transformed scales in order to plot them correctly.
# It determines the Jacobian using symbolic differentiation if possible,
# using stats::D(). If symbolic differentation fails, it falls back
# to numericDeriv(), which is less reliable; therefore, it is
# advisable to use scale transformation functions that are defined in
# terms of basic math functions so that their derivatives can be
# determined analytically (most of the transformation functions in the
# scales package currently have this property).
# For example, here is a log-Normal distribution plotted on the log
# scale, where it will appear Normal:
data.frame(dist = "lnorm", logmean = log(10), logsd = 2*log(10)) %>%
  ggplot(aes(y = 1, dist = dist, arg1 = logmean, arg2 = logsd)) +
  stat_halfeye() +
  scale_x_log10(breaks = 10^seq(-5,7, by = 2))

# see vignette("slabinterval") for many more examples.
```

`stat_spike` *Spike plot (ggplot2 stat)*

Description

Stat for drawing "spikes" (optionally with points on them) at specific points on a distribution (numerical or determined as a function of the distribution), intended for annotating `stat_slabinterval()` geometries.

Usage

```
stat_spike(
  mapping = NULL,
  data = NULL,
  geom = "spike",
  position = "identity",
  ...,
  at = "median",
  p_limits = c(NA, NA),
  density = "bounded",
  adjust = waiver(),
  trim = waiver(),
  breaks = waiver(),
```

```
  align = waiver(),
  outline_bars = waiver(),
  expand = FALSE,
  limits = NULL,
  n = waiver(),
  orientation = NA,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping` Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

`data` The data to be displayed in this layer. There are three options:

If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.

A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.

A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).

`geom` <Geom | string> Use to override the default connection between `stat_spike()` and `geom_spike()`

`position` <Position | string> Position adjustment, either as a string, or the result of a call to a position adjustment function. Setting this equal to `"dodge"` (`position_dodge()`) or `"dodgejust"` (`position_dodgejust()`) can be useful if you have overlapping geometries.

`...` Other arguments passed to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `linewidth = 3` (see **Aesthetics**, below). They may also be parameters to the paired geom/stat. When paired with the default geom, `geom_spike()`, these include:

`subguide` <function | string> Sub-guide used to annotate the `thickness` scale. One of:

- A function that takes a `scale` argument giving a ggplot2::Scale object and an `orientation` argument giving the orientation of the geometry and then returns a grid::grob that will draw the axis annotation, such as `subguide_axis()` (to draw a traditional axis) or `subguide_none()` (to draw no annotation). See `subguide_axis()` for a list of possibilities and examples.
- A string giving the name of such a function when prefixed with `"subguide_"`; e.g. `"axis"` or `"none"`. The values `"slab"`, `"dots"`, and `"spike"` use the default subguide for their geom families (no subguide), which can

be modified by setting subguide_slab, subguide_dots, or subguide_spike; see the documentation for those functions.

subscale <function | string> Sub-scale used to scale values of the thickness aesthetic within the groups determined by normalize. One of:

- A function that takes an x argument giving a numeric vector of values to be scaled and then returns a thickness vector representing the scaled values, such as subscale_thickness() or subscale_identity().
- A string giving the name of such a function when prefixed with "subscale_"; e.g. "thickness" or "identity". The value "thickness" using the default subscale, which can be modified by setting subscale_thickness; see the documentation for that function.

For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.

normalize <string> Groups within which to scale values of the thickness aesthetic. One of:

- "all": normalize so that the maximum height across all data is 1.
- "panels": normalize within panels so that the maximum height in each panel is 1.
- "xy": normalize within the x/y axis opposite the orientation of this geom so that the maximum height at each value of the opposite axis is 1.
- "groups": normalize within values of the opposite axis and within each group so that the maximum height in each group is 1.
- "none": values are taken as is with no normalization (this should probably only be used with functions whose values are in [0,1], such as CDFs).

For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.

arrow <arrow | NULL> Type of arrow heads to use on the spike, or NULL for no arrows.

at <numeric | function | character | list> The points at which to evaluate the PDF and CDF of the distribution. One of:

- numeric vector: points to evaluate the PDF and CDF of the distributions at.
- function or character vector: function (or names of functions) which, when applied on a distribution-like object (e.g. a **distributional** object or a posterior::rvar()), returns a vector of values to evaluate the distribution functions at.
- a list where each element is any of the above (e.g. a numeric, function, or name of a function): the evaluation points determined by each element of the list are concatenated together. This means, e.g., c(0, median, qi) would add a spike at 0, the median, and the endpoints of the qi of the distribution.

The values of at are also converted into a character vector which is supplied as a computed variable (also called at) generated by this stat, which can be mapped onto aesthetics using after_stat(). Non-empty names can be used to override the values of the computed variable; e.g. at = c(zero = 0, "median", mode = "Mode") will generate a computed variable with the values c("zero", "median", "mode") that is evaluated at 0, the median, and the mode of the distribution.

p_limits
: <length-2 numeric> Probability limits. Used to determine the lower and upper limits of *analytical* distributions (distributions from *samples* ignore this parameter and determine their limits based on the limits of the sample and the value of the trim parameter). E.g., if this is c(.001, .999), then a slab is drawn for the distribution from the quantile at p = .001 to the quantile at p = .999. If the lower (respectively upper) limit is NA, then the lower (upper) limit will be the minimum (maximum) of the distribution's support if it is finite, and 0.001 (0.999) if it is not finite. E.g., if p_limits is c(NA, NA), on a gamma distribution the effective value of p_limits would be c(0, .999) since the gamma distribution is defined on (0, Inf); whereas on a normal distribution it would be equivalent to c(.001, .999) since the normal distribution is defined on (-Inf, Inf).

density
: <function | string> Density estimator for sample data. One of:
  - A function which takes a numeric vector and returns a list with elements x (giving grid points for the density estimator) and y (the corresponding densities). **ggdist** provides a family of functions following this format, including density_unbounded() and density_bounded(). This format is also compatible with stats::density().
  - A string giving the suffix of a function name that starts with "density_"; e.g. "bounded" for [density_bounded()], "unbounded" for [density_unbounded()], or "histogram" for density_histogram(). Defaults to "bounded", i.e. density_bounded(), which estimates the bounds from the data and then uses a bounded density estimator based on the reflection method.

adjust
: <scalar numeric | waiver> Passed to density (e.g. density_bounded()): Value to multiply the bandwidth of the density estimator by. Default waiver() defers to the default of the density estimator, which is usually 1.

trim
: <scalar logical | waiver> Passed to density (e.g. density_bounded()): Should the density estimate be trimmed to the range of the data? Default waiver() defers to the default of the density estimator, which is usually TRUE.

breaks
: <numeric | function | string | waiver> Passed to density (e.g. density_histogram()): Determines the breakpoints defining bins. Default waiver() defers to the default of the density estimator, which is usually "Scott". Similar to (but not exactly the same as) the breaks argument to graphics::hist(). One of:
  - A scalar (length-1) numeric giving the number of bins
  - A vector numeric giving the breakpoints between histogram bins
  - A function taking x and weights and returning either the number of bins or a vector of breakpoints
  - A string giving the suffix of a function that starts with "breaks_". **ggdist** provides weighted implementations of the "Sturges", "Scott", and "FD" break-finding algorithms from graphics::hist(), as well as breaks_fixed() for manually setting the bin width. See breaks.

  For example, breaks = "Sturges" will use the breaks_Sturges() algorithm, breaks = 9 will create 9 bins, and breaks = breaks_fixed(width = 1) will set the bin width to 1.

align — <scalar numeric | function | string | waiver> Passed to density (e.g. density_histogram()): Determines how to align the breakpoints defining bins. Default waiver() defers to the default of the density estimator, which is usually "none" (performs no alignment). One of:

- A scalar (length-1) numeric giving an offset that is subtracted from the breaks. The offset must be between 0 and the bin width.
- A function taking a sorted vector of breaks (bin edges) and returning an offset to subtract from the breaks.
- A string giving the suffix of a function that starts with "align_" used to determine the alignment, such as align_none(), align_boundary(), or align_center().

For example, align = "none" will provide no alignment, align = align_center(at = 0) will center a bin on 0, and align = align_boundary(at = 0) will align a bin edge on 0.

outline_bars — <scalar logical | waiver> Passed to density (e.g. density_histogram()) and also used for discrete analytical distributions (whose slabs are drawn as histograms). Determines if outlines in between the bars are drawn. If waiver() or FALSE (the default), the outline is drawn only along the tops of the bars. If TRUE, outlines in between bars are also drawn (though you may have to set the slab_color or color aesthetic to see the outlines).

expand — <logical> For sample data, should the slab be expanded to the limits of the scale? Default FALSE. Can be a length-two logical vector to control expansion to the lower and upper limit respectively.

limits — <length-2 numeric> 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 — <scalar numeric> Number of points at which to evaluate the function that defines the slab. Also passed to density (e.g. density_bounded()). Default waiver() uses the value 501 for analytical distributions and defers to the default of the density estimator for sample-based distributions, which is also usually 501.

orientation — <string> Whether this geom is drawn horizontally or vertically. One of:

- NA (default): automatically detect the orientation based on how the aesthetics are assigned. Automatic detection works most of the time.
- "horizontal" (or "y"): draw horizontally, using the y aesthetic to identify different groups. For each group, uses the x, xmin, xmax, and thickness aesthetics to draw points, intervals, and slabs.
- "vertical" (or "x"): draw vertically, using the x aesthetic to identify different groups. For each group, uses the y, ymin, ymax, and thickness aesthetics to draw points, intervals, and slabs.

For compatibility with the base ggplot naming scheme for orientation, "x" can be used as an alias for "vertical" and "y" as an alias for "horizontal" (**ggdist** had an orientation parameter before base ggplot did, hence the discrepancy).

na.rm <scalar logical> 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? 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). It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders().

Details

This stat computes slab values (i.e. PDF and CDF values) at specified locations on a distribution, as determined by the at parameter.

To visualize sample data, such as a data distribution, samples from a bootstrap distribution, or a Bayesian posterior, you can supply samples to the x or y aesthetic.

To visualize analytical distributions, you can use the xdist or ydist aesthetic. For historical reasons, you can also use dist to specify the distribution, though this is not recommended as it does not work as well with orientation detection. These aesthetics can be used as follows:

- xdist, ydist, and dist can be any distribution object from the distributional package (dist_normal(), dist_beta(), etc) or can be a posterior::rvar() object. Since these functions are vectorized, other columns can be passed directly to them in an aes() specification; e.g. aes(dist = dist_normal(mu, sigma)) will work if mu and sigma are columns in the input data frame.
- dist can be a character vector giving the distribution name. Then the arg1, ... arg9 aesthetics (or args as a list column) specify distribution arguments. Distribution names should correspond to R functions that have "p", "q", and "d" functions; e.g. "norm" is a valid distribution name because R defines the pnorm(), qnorm(), and dnorm() functions for Normal distributions.

  See the parse_dist() function for a useful way to generate dist and args values from human-readable distribution specs (like "normal(0,1)"). Such specs are also produced by other packages (like the brms::get_prior function in brms); thus, parse_dist() combined with the stats described here can help you visualize the output of those functions.

Value

A ggplot2::Stat representing a spike geometry which can be added to a ggplot() object.

Aesthetics

The spike geom has a wide variety of aesthetics that control the appearance of its two sub-geometries: the **spike** and the **point**.

These stats support the following aesthetics:

- x: x position of the geometry (when orientation = "vertical"); or sample data to be summarized (when orientation = "horizontal" with sample data).
- y: y position of the geometry (when orientation = "horizontal"); or sample data to be summarized (when orientation = "vertical" with sample data).
- weight: When using samples (i.e. the x and y aesthetics, not xdist or ydist), optional weights to be applied to each draw.
- xdist: When using analytical distributions, distribution to map on the x axis: a **distributional** object (e.g. dist_normal()) or a posterior::rvar() object.
- ydist: When using analytical distributions, distribution to map on the y axis: a **distributional** object (e.g. dist_normal()) or a posterior::rvar() object.
- dist: When using analytical distributions, a name of a distribution (e.g. "norm"), a **distributional** object (e.g. dist_normal()), or a posterior::rvar() object. See **Details**.
- args: Distribution arguments (args or arg1, ... arg9). See **Details**.

In addition, in their default configuration (paired with geom_spike()) the following aesthetics are supported by the underlying geom:

Spike-specific (aka Slab-specific) aesthetics

- thickness: The thickness of the slab at each x value (if orientation = "horizontal") or y value (if orientation = "vertical") of the slab.
- side: Which side to place the slab on. "topright", "top", and "right" are synonyms which cause the slab to be drawn on the top or the right depending on if orientation is "horizontal" or "vertical". "bottomleft", "bottom", and "left" are synonyms which cause the slab to be drawn on the bottom or the left depending on if orientation is "horizontal" or "vertical". "topleft" causes the slab to be drawn on the top or the left, and "bottomright" causes the slab to be drawn on the bottom or the right. "both" draws the slab mirrored on both sides (as in a violin plot).
- scale: What proportion of the region allocated to this geom to use to draw the slab. If scale = 1, slabs that use the maximum range will just touch each other. Default is 0.9 to leave some space between adjacent slabs. For a comprehensive discussion and examples of slab scaling and normalization, see the thickness scale article.

Color aesthetics

- colour: (or color) The color of the **spike** and **point** sub-geometries.
- fill: The fill color of the **point** sub-geometry.
- alpha: The opacity of the **spike** and **point** sub-geometries.
- colour_ramp: (or color_ramp) A secondary scale that modifies the color scale to "ramp" to another color. See scale_colour_ramp() for examples.
- fill_ramp: A secondary scale that modifies the fill scale to "ramp" to another color. See scale_fill_ramp() for examples.

Line aesthetics

- linewidth: Width of the line used to draw the **spike** sub-geometry.

- `size`: Size of the **point** sub-geometry.
- `stroke`: Width of the outline around the **point** sub-geometry.
- `linetype`: Type of line (e.g., `"solid"`, `"dashed"`, etc) used to draw the **spike**.

Other aesthetics (these work as in standard geoms)

- `width`
- `height`
- `group`

See examples of some of these aesthetics in action in `vignette("slabinterval")`. Learn more about the sub-geom override aesthetics (like `interval_color`) in the scales documentation. Learn more about basic ggplot aesthetics in `vignette("ggplot2-specs")`.

Computed Variables

The following variables are computed by this stat and made available for use in aesthetic specifications (`aes()`) using the `after_stat()` function or the `after_stat` argument of `stage()`:

- `x` or `y`: For slabs, the input values to the slab function. For intervals, the point summary from the interval function. Whether it is `x` or `y` depends on `orientation`
- `xmin` or `ymin`: For intervals, the lower end of the interval from the interval function.
- `xmax` or `ymax`: For intervals, the upper end of the interval from the interval function.
- `.width`: For intervals, the interval width as a numeric value in `[0, 1]`. For slabs, the width of the smallest interval containing that value of the slab.
- `level`: For intervals, the interval width as an ordered factor. For slabs, the level of the smallest interval containing that value of the slab.
- `pdf`: For slabs, the probability density function (PDF). If `options("ggdist.experimental.slab_data_in_interval` is `TRUE`: For intervals, the PDF at the point summary; intervals also have `pdf_min` and `pdf_max` for the PDF at the lower and upper ends of the interval.
- `cdf`: For slabs, the cumulative distribution function. If `options("ggdist.experimental.slab_data_in_intervals'` is `TRUE`: For intervals, the CDF at the point summary; intervals also have `cdf_min` and `cdf_max` for the CDF at the lower and upper ends of the interval.
- `n`: For slabs, the number of data points summarized into that slab. If the slab was created from an analytical distribution via the `xdist`, `ydist`, or `dist` aesthetic, `n` will be `Inf`.
- `f`: (deprecated) For slabs, the output values from the slab function (such as the PDF, CDF, or CCDF), determined by `slab_type`. Instead of using `slab_type` to change `f` and then mapping `f` onto an aesthetic, it is now recommended to simply map the corresponding computed variable (e.g. `pdf`, `cdf`, or `1 - cdf`) directly onto the desired aesthetic.
- `at`: For spikes, a character vector of names of the functions or expressions used to determine the points at which the slab functions were evaluated to create spikes. Values of this computed variable are determined by the `at` parameter; see its description above.

See Also

See geom_spike() for the geom underlying this stat. See stat_slabinterval() for the stat this shortcut is based on.

Other slabinterval stats: stat_ccdfinterval(), stat_cdfinterval(), stat_eye(), stat_gradientinterval(), stat_halfeye(), stat_histinterval(), stat_interval(), stat_pointinterval(), stat_slab()

Examples

```
library(ggplot2)
library(distributional)
library(dplyr)

df = tibble(
  d = c(dist_normal(1), dist_gamma(2,2)), g = c("a", "b")
)

# annotate the density at the mode of a distribution
df %>%
  ggplot(aes(y = g, xdist = d)) +
  stat_slab(aes(xdist = d)) +
  stat_spike(at = "Mode") +
  # need shared thickness scale so that stat_slab and geom_spike line up
  scale_thickness_shared()

# annotate the endpoints of intervals of a distribution
# here we'll use an arrow instead of a point by setting size = 0
arrow_spec = arrow(angle = 45, type = "closed", length = unit(4, "pt"))
df %>%
  ggplot(aes(y = g, xdist = d)) +
  stat_halfeye(point_interval = mode_hdci) +
  stat_spike(
    at = function(x) hdci(x, .width = .66),
    size = 0, arrow = arrow_spec, color = "blue", linewidth = 0.75
  ) +
  scale_thickness_shared()

# annotate quantiles of a sample
set.seed(1234)
data.frame(x = rnorm(1000, 1:2), g = c("a","b")) %>%
  ggplot(aes(x, g)) +
  stat_slab() +
  stat_spike(at = function(x) quantile(x, ppoints(10))) +
  scale_thickness_shared()
```

---

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, q	vector of quantiles.
df	degrees of freedom (> 0, maybe non-integer). df = Inf is allowed.
mu	<numeric> Location parameter (median).
sigma	<numeric> Scale parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
p	vector of probabilities.
n	number of observations. If length(n) > 1, the length is taken to be the number required.

Value

- dstudent_t gives the density
- pstudent_t gives the cumulative distribution function (CDF)
- qstudent_t gives the quantile function (inverse CDF)
- rstudent_t generates random draws.

The length of the result is determined by n for rstudent_t, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

See Also

parse_dist() and parsing distribution specs and the stat_slabinterval() family of stats for visualizing them.

Examples

```
library(dplyr)
library(ggplot2)

expand.grid(
  df = c(3,5,10,30),
  scale = c(1,1.5)
) %>%
 ggplot(aes(y = 0, dist = "student_t", arg1 = df, arg2 = 0, arg3 = scale, color = ordered(df))) +
  stat_slab(p_limits = c(.01, .99), fill = NA) +
  scale_y_continuous(breaks = NULL) +
  facet_grid( ~ scale) +
  labs(
    title = "dstudent_t(x, df, 0, sigma)",
    subtitle = "Scale (sigma)",
    y = NULL,
    x = NULL
  ) +
  theme_ggdist() +
  theme(axis.title = element_text(hjust = 0))
```

`sub-geometry-scales` *Sub-geometry scales for geom_slabinterval (ggplot2 scales)*

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 = guide_colourbar2()
)

scale_point_color_continuous(
  ...,
  aesthetics = "point_colour",
  guide = guide_colourbar2()
)
```

```
scale_point_fill_discrete(..., aesthetics = "point_fill")

scale_point_fill_continuous(
  ...,
  aesthetics = "point_fill",
  guide = guide_colourbar2()
)

scale_point_alpha_continuous(..., range = c(0.1, 1))

scale_point_alpha_discrete(..., range = c(0.1, 1))

scale_point_size_continuous(..., range = c(1, 6))

scale_point_size_discrete(..., range = c(1, 6), na.translate = FALSE)

scale_interval_colour_discrete(..., aesthetics = "interval_colour")

scale_interval_color_discrete(..., aesthetics = "interval_colour")

scale_interval_colour_continuous(
  ...,
  aesthetics = "interval_colour",
  guide = guide_colourbar2()
)

scale_interval_color_continuous(
  ...,
  aesthetics = "interval_colour",
  guide = guide_colourbar2()
)

scale_interval_alpha_continuous(..., range = c(0.1, 1))

scale_interval_alpha_discrete(..., range = c(0.1, 1))

scale_interval_size_continuous(..., range = c(1, 6))

scale_interval_size_discrete(..., range = c(1, 6), na.translate = FALSE)

scale_interval_linetype_discrete(..., na.value = "blank")

scale_interval_linetype_continuous(...)

scale_slab_colour_discrete(..., aesthetics = "slab_colour")

scale_slab_color_discrete(..., aesthetics = "slab_colour")
```

```
scale_slab_colour_continuous(
  ...,
  aesthetics = "slab_colour",
  guide = guide_colourbar2()
)

scale_slab_color_continuous(
  ...,
  aesthetics = "slab_colour",
  guide = guide_colourbar2()
)

scale_slab_fill_discrete(..., aesthetics = "slab_fill")

scale_slab_fill_continuous(
  ...,
  aesthetics = "slab_fill",
  guide = guide_colourbar2()
)

scale_slab_alpha_continuous(
  ...,
  limits = function(l) c(min(0, l[[1]]), l[[2]]),
  range = c(0, 1)
)

scale_slab_alpha_discrete(..., range = c(0.1, 1))

scale_slab_size_continuous(..., range = c(1, 6))

scale_slab_size_discrete(..., range = c(1, 6), na.translate = FALSE)

scale_slab_linewidth_continuous(..., range = c(1, 6))

scale_slab_linewidth_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` <character> Names of aesthetics to set scales for.

`guide` <Guide | string> Guide to use for legends for an aesthetic.

`range` <length-2 numeric> The minimum and maximum size of the plotting symbol after transformation.

`na.translate` <scalar logical> In discrete scales, should we show missing values?

`na.value` <linetype> When `na.translate` is `TRUE`, what value should be shown?

`limits` One of:

- `NULL` to use the default scale range
- A numeric vector of length two providing limits of the scale. Use `NA` to refer to the existing minimum or maximum
- A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang lambda function notation. Note that setting limits on positional scales will **remove** data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see `coord_cartesian()`).

`solid` Should the shapes be solid, `TRUE`, or hollow, `FALSE`?

Details

The following additional scales / aesthetics are defined for use with `geom_slabinterval()` and related geoms:

`scale_point_color_*` Point color

`scale_point_fill_*` Point fill color

`scale_point_alpha_*` Point alpha level / opacity

`scale_point_size_*` Point size

`scale_interval_color_*` Interval line color

`scale_interval_alpha_*` Interval alpha level / opacity

`scale_interval_linetype_*` Interval line type

`scale_slab_color_*` Slab outline color

`scale_slab_fill_*` Slab fill color

`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.

`scale_slab_linewidth_*` Slab outline line width

`scale_slab_linetype_*` Slab outline line type

`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")`

These scales have been deprecated:

`scale_interval_size_*`	Use `scale_linewidth_*`
`scale_slab_size_*`	Slab `scale_size_linewidth_*`

Value

A ggplot2::Scale representing one of the aesthetics used to target the appearance of specific parts of composite `ggdist` geoms. Can be added to a `ggplot()` object.

Author(s)

Matthew Kay

See Also

Other ggplot2 scales: `scale_color_discrete()`, `scale_color_continuous()`, etc.

Other ggdist scales: `scale_colour_ramp`, `scale_side_mirrored()`, `scale_thickness`

Examples

```
library(dplyr)
library(ggplot2)

# This plot shows how to set multiple specific aesthetics
# NB it is very ugly and is only for demo purposes.
data.frame(distribution = "Normal(1,2)") %>%
  parse_dist(distribution) %>%
  ggplot(aes(y = distribution, xdist = .dist, args = .args)) +
  stat_halfeye(
    shape = 21,  # this point shape has a fill and outline
    point_color = "red",
    point_fill = "black",
    point_alpha = .1,
    point_size = 6,
    stroke = 2,
    interval_color = "blue",
    # interval line widths are scaled from [1, 6] onto [0.6, 1.4] by default
    # see the interval_size_range parameter in help("geom_slabinterval")
    linewidth = 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_linewidth = 3,
    slab_linetype = "dotted",
    slab_alpha = .5
  )
```

---

subguide_axis *Axis sub-guide for thickness scales*

Description

This is a sub-guide intended for annotating the `thickness` and dot-count subscales in **ggdist**. It can be used with the `subguide` parameter of `geom_slabinterval()` and `geom_dotsinterval()`.

Supports automatic partial function application with waived arguments.

Usage

```
subguide_axis(
  values,
  title = NULL,
  breaks = waiver(),
  labels = waiver(),
  position = 0,
  just = 0,
  label_side = "topright",
  orientation = "horizontal",
  theme = theme_get()
)

subguide_inside(..., label_side = "inside")

subguide_outside(..., label_side = "outside", just = 1)

subguide_integer(..., breaks = scales::breaks_extended(Q = c(1, 5, 2, 4, 3)))

subguide_count(..., breaks = scales::breaks_width(1))

subguide_slab(values, ...)

subguide_dots(values, ...)

subguide_spike(values, ...)
```

Arguments

`values`	<numeric> Values used to construct the scale used for this guide. Typically provided automatically by `geom_slabinterval()`.
`title`	<string> The title of the scale shown on the sub-guide's axis.
`breaks`	One of: • `NULL` for no breaks • `waiver()` for the default breaks computed by the transformation object • A numeric vector of positions • A function that takes the limits as input and returns breaks as output (e.g., a function returned by `scales::extended_breaks()`). Note that for position scales, limits are provided after scale expansion. Also accepts rlang lambda function notation.
`labels`	One of: • `NULL` for no labels • `waiver()` for the default labels computed by the transformation object • A character vector giving labels (must be same length as `breaks`) • An expression vector (must be the same length as breaks). See ?plotmath for details. • A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.
`position`	<scalar numeric> Value between `0` and `1` giving the position of the guide relative to the axis: `0` causes the sub-guide to be drawn on the left or bottom depending on if `orientation` is `"horizontal"` or `"vertical"`, and `1` causes the sub-guide to be drawn on the top or right depending on if `orientation` is `"horizontal"` or `"vertical"`. May also be a string indicating the position: `"top"`, `"right"`, `"bottom"`, `"left"`, `"topright"`, `"topleft"`, `"bottomright"`, or `"bottomleft"`.
`just`	<scalar numeric> Value between `0` and `1` giving the justification of the guide relative to its position: 0 means aligned towards the inside of the axis edge, 1 means aligned towards the outside of the axis edge.
`label_side`	<string> Which side of the axis to draw the ticks and labels on. `"topright"`, `"top"`, and `"right"` are synonyms which cause the labels 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 labels to be drawn on the bottom or the left depending on if `orientation` is `"horizontal"` or `"vertical"`. `"topleft"` causes the labels to be drawn on the top or the left, and `"bottomright"` causes the labels to be drawn on the bottom or the right. `"inside"` causes the labels to be drawn on the side closest to the inside of the chart, depending on `position`, and `"outside"` on the side closest to the outside of the chart.
`orientation`	<string> Orientation of the geometry this sub-guide is for. One of `"horizontal"` (`"y"`) or `"vertical"` (`"x"`). See the `orientation` parameter to `geom_slabinterval()`.
`theme`	<theme> Theme used to determine the style that the sub-guide elements are drawn in. The title label is drawn using the `"axis.title.x"` or `"axis.title.y"` theme setting, and the axis line, ticks, and tick labels are drawn using `guide_axis()`, so the same theme settings that normally apply to axis guides will be followed.

... Arguments passed to other functions, typically back to subguide_axis() itself.

Details

subguide_inside() is a shortcut for drawing labels inside of the chart region.

subguide_outside() is a shortcut for drawing labels outside of the chart region.

subguide_integer() only draws breaks that are integer values, useful for labeling counts in geom_dots().

subguide_count() is a shortcut for drawing labels where *every* whole number is labeled, useful for labeling counts in geom_dots(). If your max count is large, subguide_integer() may be better.

subguide_slab(), subguide_dots(), and subguide_spike() are aliases for subguide_none() that allow you to change the default subguide used for the geom_slabinterval(), geom_dotsinterval(), and geom_spike() families. If you overwrite these in the global environment, you can set the corresponding default subguide. For example:

```
subguide_slab = ggdist::subguide_inside(position = "right")
```

This will cause geom_slabinterval()s to default to having a guide on the right side of the geom.

See Also

The thickness datatype.

The thickness aesthetic of geom_slabinterval().

scale_thickness_shared(), for setting a thickness scale across all geometries using the thickness aesthetic.

subscale_thickness(), for setting a thickness sub-scale within a single geom_slabinterval().

Other sub-guides: subguide_none()

Examples

```
library(ggplot2)
library(distributional)

df = data.frame(d = dist_normal(2:3, 2:3), g = c("a", "b"))

# subguides allow you to label thickness axes
ggplot(df, aes(xdist = d, y = g)) +
  stat_slabinterval(subguide = "inside")

# they respect normalization and use of scale_thickness_shared()
ggplot(df, aes(xdist = d, y = g)) +
  stat_slabinterval(subguide = "inside", normalize = "groups")

# they can also be positioned outside the plot area, though
# this typically requires manually adjusting plot margins
ggplot(df, aes(xdist = d, y = g)) +
  stat_slabinterval(subguide = subguide_outside(title = "density", position = "right")) +
  theme(plot.margin = margin(5.5, 50, 5.5, 5.5))
```

```
# any of the subguide types will also work to indicate bin counts in
# geom_dots(); subguide_integer() and subguide_count() can be useful for
# dotplots as they only label integers / whole numbers:
df = data.frame(d = dist_gamma(2:3, 2:3), g = c("a", "b"))
ggplot(df, aes(xdist = d, y = g)) +
  stat_dots(subguide = subguide_count(label_side = "left", title = "count")) +
  scale_y_discrete(expand = expansion(add = 0.1)) +
  scale_x_continuous(expand = expansion(add = 0.5))
```

`subguide_none` *Empty sub-guide for thickness scales*

Description

This is a blank sub-guide that omits annotations for the `thickness` and dot-count sub-scales in **ggdist**. It can be used with the `subguide` parameter of `geom_slabinterval()` and `geom_dotsinterval()`.

Supports automatic partial function application with waived arguments.

Usage

```
subguide_none(values, ...)
```

Arguments

`values`	<numeric> Values used to construct the scale used for this guide. Typically provided automatically by `geom_slabinterval()`.
`...`	ignored.

See Also

Other sub-guides: `subguide_axis()`

`subscale_identity` *Identity sub-scale for thickness aesthetic*

Description

This is an identity sub-scale for the `thickness` aesthetic in **ggdist**. It returns its input as a thickness vector without rescaling. It can be used with the `subscale` parameter of `geom_slabinterval()`.

Usage

```
subscale_identity(x)
```

Arguments

`x`	<numeric> Vector to be rescaled. Typically provided automatically by `geom_slabinterval()`.

Value

A thickness vector of the same length as `x`, with infinite values in `x` squished into the data range.

See Also

Other sub-scales: `subscale_thickness()`

`subscale_thickness` *Sub-scale for thickness aesthetic*

Description

This is a sub-scale intended for adjusting the scaling of the `thickness` aesthetic at a geometry (or sub-geometry) level in **ggdist**. It can be used with the `subscale` parameter of `geom_slabinterval()`.

Supports automatic partial function application with waived arguments.

Usage

```
subscale_thickness(
  x,
  limits = function(l) c(min(0, l[1]), l[2]),
  expand = c(0, 0)
)
```

Arguments

`x` <numeric> Vector to be rescaled. Typically provided automatically by `geom_slabinterval()`.

`limits` <length-2 numeric | function | NULL> One of:

- A numeric vector of length two providing the limits of the scale. Use `NA` to use the default minimum or maximum.
- A function that accepts a length-2 numeric vector of the automatic limits and returns new limits. Unlike positional scales, these limits will not remove data.
- `NULL` to use the range of the data

`expand` <numeric> Vector of limit expansion constants of length 2 or 4, following the same format used by the `expand` argument of `continuous_scale()`. The default is not to expand the limits. You can use the convenience function `expansion()` to generate the expansion values; expanding the lower limit is usually not recommended (because with most `thickness` scales the lower limit is the baseline and represents `0`), so a typical usage might be something like `expand = expansion(c(0, 0.05))` to expand the top end of the scale by 5%.

Details

You can overwrite `subscale_thickness` in the global environment to set the default properties of the thickness subscale. For example:

```
subscale_thickness = ggdist::subscale_thickness(expand = expansion(c(0, 0.05)))
```

This will cause `geom_slabinterval()`s to default to a thickness subscale that expands by 5% at the top of the scale. **Always** prefix such a definition with `ggdist::` to avoid infinite loops caused by recursion.

Value

A thickness vector of the same length as `x` scaled to be between `0` and `1`.

See Also

The thickness datatype.

The `thickness` aesthetic of `geom_slabinterval()`.

`scale_thickness_shared()`, for setting a `thickness` scale across all geometries using the `thickness` aesthetic.

Other sub-scales: `subscale_identity()`

Examples

```
library(ggplot2)
library(distributional)

df = data.frame(d = dist_normal(2:3, 1), g = c("a", "b"))

# breaks on thickness subguides are always limited to the bounds of the
# subscale, which may leave labels off near the edge of the subscale
# (e.g. here `0.4` is omitted because the max value is approx `0.39`)
ggplot(df, aes(xdist = d, y = g)) +
  stat_slabinterval(
    subguide = "inside"
  )

# We can use the subscale to expand the upper limit of the thickness scale
# by 5% (similar to the default for positional scales), allowing bounds near
# (but just less than) the limit, like `0.4`, to be shown.
ggplot(df, aes(xdist = d, y = g)) +
  stat_slabinterval(
    subguide = "inside",
    subscale = subscale_thickness(expand = expansion(c(0, 0.5)))
  )
```

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(
  base_size = 11,
  base_family = "",
  base_line_size = base_size/22,
  base_rect_size = base_size/22
)

theme_tidybayes(
  base_size = 11,
  base_family = "",
  base_line_size = base_size/22,
  base_rect_size = base_size/22
)

facet_title_horizontal()

axis_titles_bottom_left()

facet_title_left_horizontal()

facet_title_right_horizontal()
```

Arguments

base_size	base font size, given in pts.
base_family	base font family
base_line_size	base size for line elements
base_rect_size	base size for rect elements

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())
```

`thickness` *Thickness (datatype)*

Description

A representation of the thickness of a slab: a scaled value (`x`) where `0` is the base of the slab and `1` is its maximum extent, and the lower (`lower`) and upper (`upper`) limits of the slab values in their original data units.

Usage

```
thickness(x = double(), lower = NA_real_, upper = NA_real_)
```

Arguments

`x`	<coercible-to-numeric> A numeric vector or an object coercible to a numeric (via `vctrs::vec_cast()`) representing scaled values to be converted to a `thickness()` object.
`lower`	<numeric> The original lower bounds of thickness values before scaling. May be `NA` to indicate that this bound is not known.
`upper`	<numeric> The original upper bounds of thickness values before scaling. May be `NA` to indicate that this bound is not known.

Details

This datatype is used by `scale_thickness_shared()` and `subscale_thickness()` to represent `numeric()`-like objects marked as being in units of slab "thickness".

Unlike regular `numeric()`s, `thickness()` values mapped onto the `thickness` aesthetic are not rescaled by `scale_thickness_shared()` or `geom_slabinterval()`. In most cases `thickness()` is not useful directly; though it can be used to mark values that should not be rescaled—see the definitions of `stat_ccdfinterval()` and `stat_gradientinterval()` for some example usages.

thickness objects with unequal lower or upper limits may not be combined. However, thickness objects with `NA` limits may be combined with thickness objects with non-`NA` limits. This allows (e.g.) specifying locations on the thickness scale that are independent of data limits.

Value

A vctrs::rcrd of class `"ggdist_thickness"` with fields `"x"`, `"lower"`, and `"upper"`.

Author(s)

Matthew Kay

See Also

The `thickness` aesthetic of `geom_slabinterval()`.

`scale_thickness_shared()`, for setting a `thickness` scale across all geometries using the `thickness` aesthetic.

`subscale_thickness()`, for setting a `thickness` sub-scale within a single `geom_slabinterval()`.

Examples

```
thickness(0:1)
thickness(0:1, 0, 10)
```

`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`	<data.frame> 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()
```

---

waiver *A waived argument*

Description

A flag indicating that the default value of an argument should be used.

Usage

```
waiver()
```

Details

A `waiver()` is a flag passed to a function argument that indicates the function should use the default value of that argument. It is used in two cases:

- **ggplot2** functions use it to distinguish between "nothing" (`NULL`) and a default value calculated elsewhere (`waiver()`).
- **ggdist** turns **ggplot2**'s convention into a standardized method of argument-passing: any named argument with a default value in an automatically partially-applied function can be passed `waiver()` when calling the function. This will cause the default value (or the most recently partially-applied value) of that argument to be used instead.

  Note: due to historical limitations, `waiver()` cannot currently be used on arguments to the `point_interval()` family of functions.

See Also

`auto_partial()`, `ggplot2::waiver()`

Examples

```
f = auto_partial(function(x, y = "b") {
  c(x = x, y = y)
})

f("a")

# uses the default value of `y` ("b")
f("a", y = waiver())

# partially apply `f`
g = f(y = "c")
g

# uses the last partially-applied value of `y` ("c")
g("a", y = waiver())
```

---

weighted_ecdf — *Weighted empirical cumulative distribution function*

Description

A variation of ecdf() that can be applied to weighted samples.

Usage

```
weighted_ecdf(x, weights = NULL, na.rm = FALSE)
```

Arguments

x	<numeric> Sample values.
weights	<numeric | NULL> Weights for the sample. One of: • numeric vector of same length as x: weights for corresponding values in x, which will be normalized to sum to 1. • NULL: indicates no weights are provided, so the unweighted empirical cumulative distribution function (equivalent to ecdf()) is returned.
na.rm	<scalar logical> If TRUE, corresponding entries in x and weights are removed if either is NA.

Details

Generates a weighted empirical cumulative distribution function, $F(x)$. Given x, a sorted vector (derived from x), and w_i, the corresponding weight for x_i, $F(x)$ is a step function with steps at each x_i with $F(x_i)$ equal to the sum of all weights up to and including w_i.

Value

weighted_ecdf() returns a function of class "weighted_ecdf", which also inherits from the stepfun() class. Thus, it also has plot() and print() methods. Like ecdf(), weighted_ecdf() also provides a quantile() method, which dispatches to weighted_quantile().

See Also

weighted_quantile()

Examples

```
weighted_ecdf(1:3, weights = 1:3)
plot(weighted_ecdf(1:3, weights = 1:3))
quantile(weighted_ecdf(1:3, weights = 1:3), 0.4)
```

---

weighted_quantile *Weighted sample quantiles*

Description

A variation of quantile() that can be applied to weighted samples.

Usage

```
weighted_quantile(
  x,
  probs = seq(0, 1, 0.25),
  weights = NULL,
  n = NULL,
  na.rm = FALSE,
  names = TRUE,
  type = 7,
  digits = 7
)

weighted_quantile_fun(x, weights = NULL, n = NULL, na.rm = FALSE, type = 7)
```

Arguments

x	<numeric> Sample values.
probs	<numeric> Vector of probabilities in $[0, 1]$ defining the quantiles to return.
weights	<numeric | NULL> Weights for the sample. One of: • numeric vector of same length as x: weights for corresponding values in x, which will be normalized to sum to 1. • NULL: indicates no weights are provided, so unweighted sample quantiles (equivalent to quantile()) are returned.

n <scalar numeric> Presumed effective sample size. If this is greater than 1 and continuous quantiles (type >= 4) are requested, flat regions may be added to the approximation to the inverse CDF in areas where the normalized weight exceeds 1/n (i.e., regions of high density). This can be used to ensure that if a sample of size n with duplicate x values is summarized into a weighted sample without duplicates, the result of weighted_quantile(..., n = n) on the weighted sample is equal to the result of quantile() on the original sample. One of:

- NULL: do not make a sample size adjustment.
- numeric: presumed effective sample size.
- function or name of function (as a string): A function applied to weights (prior to normalization) to determine the sample size. Some useful values may be:
  - "length": i.e. use the number of elements in weights (equivalently in x) as the effective sample size.
  - "sum": i.e. use the sum of the unnormalized weights as the sample size. Useful if the provided weights is unnormalized so that its sum represents the true sample size.

na.rm <scalar logical> If TRUE, corresponding entries in x and weights are removed if either is NA.

names <scalar logical> If TRUE, add names to the output giving the input probs formatted as a percentage.

type <scalar integer> Value between 1 and 9: determines the type of quantile estimator to be used. Types 1 to 3 are for discontinuous quantiles, types 4 to 9 are for continuous quantiles. See **Details**.

digits <scalar numeric> The number of digits to use to format percentages when names is TRUE.

Details

Calculates weighted quantiles using a variation of the quantile types based on a generalization of quantile().

Type 1–3 (discontinuous) quantiles are directly a function of the inverse CDF as a step function, and so can be directly translated to the weighted case using the natural definition of the weighted ECDF as the cumulative sum of the normalized weights.

Type 4–9 (continuous) quantiles require some translation from the definitions in quantile(). quantile() defines continuous estimators in terms of x_k, which is the kth order statistic, and p_k, which is a function of k and n (the sample size). In the weighted case, we instead take x_k as the kth smallest value of x in the weighted sample (not necessarily an order statistic, because of the weights). Then we can re-write the formulas for p_k in terms of $F(x_k)$ (the empirical CDF at x_k, i.e. the cumulative sum of normalized weights) and $f(x_k)$ (the normalized weight at x_k), by using the fact that, in the unweighted case, $k = F(x_k) \cdot n$ and $1/n = f(x_k)$:

Type 4 $p_k = \frac{k}{n} = F(x_k)$

Type 5 $p_k = \frac{k-0.5}{n} = F(x_k) - \frac{f(x_k)}{2}$

Type 6 $p_k = \frac{k}{n+1} = \frac{F(x_k)}{1+f(x_k)}$

Type 7 $p_k = \frac{k-1}{n-1} = \frac{F(x_k)-f(x_k)}{1-f(x_k)}$

Type 8 $p_k = \frac{k-1/3}{n+1/3} = \frac{F(x_k)-f(x_k)/3}{1+f(x_k)/3}$

Type 9 $p_k = \frac{k-3/8}{n+1/4} = \frac{F(x_k)-f(x_k)\cdot 3/8}{1+f(x_k)/4}$

Then the quantile function (inverse CDF) is the piece-wise linear function defined by the points (p_k, x_k).

Value

`weighted_quantile()` returns a numeric vector of `length(probs)` with the estimate of the corresponding quantile from `probs`.

`weighted_quantile_fun()` returns a function that takes a single argument, a vector of probabilities, which itself returns the corresponding quantile estimates. It may be useful when `weighted_quantile()` needs to be called repeatedly for the same sample, re-using some pre-computation.

See Also

`weighted_ecdf()`

Index

∗ **bounds estimators**
bounder_cdf, 13
bounder_cooke, 15
bounder_range, 16
∗ **colour ramp functions**
guide_rampbar, 111
partial_colour_ramp, 120
ramp_colours, 131
scale_colour_ramp, 132
∗ **datasets**
ggdist-deprecated, 110
∗ **density estimators**
density_bounded, 24
density_histogram, 28
density_unbounded, 30
∗ **dotplot smooths**
smooth_density, 141
smooth_discrete, 143
smooth_none, 145
∗ **dotsinterval geoms**
geom_blur_dots, 34
geom_dots, 43
geom_dotsinterval, 51
geom_swarm, 93
geom_weave, 101
∗ **dotsinterval stats**
stat_dots, 167
stat_dotsinterval, 176
stat_mcse_dots, 240
∗ **ggdist scales**
scale_colour_ramp, 132
scale_side_mirrored, 134
scale_thickness, 137
sub-geometry-scales, 291
∗ **lineribbon stats**
stat_lineribbon, 235
stat_ribbon, 256
∗ **manip**
tidy-format-translators, 304
∗ **slabinterval geoms**
geom_interval, 61
geom_pointinterval, 70
geom_slab, 75
geom_spike, 88
∗ **slabinterval stats**
stat_ccdfinterval, 146
stat_cdfinterval, 156
stat_eye, 187
stat_gradientinterval, 197
stat_halfeye, 208
stat_histinterval, 218
stat_interval, 229
stat_pointinterval, 249
stat_slab, 261
stat_spike, 281
∗ **sub-guides**
subguide_axis, 296
subguide_none, 299
∗ **sub-scales**
subscale_identity, 299
subscale_thickness, 300

aes(), *35*, *44*, *52*, *57*, *62*, *66*, *70*, *75*, *81*, *89*, *94*, *102*, *147*, *152*, *157*, *162*, *163*, *167*, *172*, *177*, *182*, *183*, *188*, *193*, *198*, *203*, *204*, *208*, *214*, *219*, *224*, *229*, *231*, *232*, *236–238*, *240*, *245*, *250*, *252*, *256*, *258*, *261*, *266*, *270*, *276*, *282*, *286*, *288*
after_stat(), *129*, *130*, *152*, *163*, *172*, *183*, *193*, *204*, *214*, *224*, *232*, *238*, *245*, *252*, *258*, *266*, *276*, *283*, *288*
align, 5, *7*, *18*
align_boundary *(*align*)*, 5
align_boundary(), *5*, *29*, *150*, *161*, *191*, *202*, *212*, *222*, *264*, *274*, *285*
align_center *(*align*)*, 5
align_center(), *5*, *29*, *150*, *161*, *191*, *202*, *212*, *222*, *264*, *274*, *285*

align_none (align), 5
align_none(), 5, 29, 150, 161, 191, 202, 212, 222, 264, 274, 285
arrow, 55, 63, 72, 84, 90, 148, 159, 180, 190, 200, 210, 221, 230, 251, 272, 283
auto_partial, 7
auto_partial(), 7, 306
automatic partial function application, 5, 9, 12, 13, 15, 16, 25, 28, 30, 121, 141, 143, 145, 296, 299, 300
automatic-partial-functions, 142, 144, 145
automatic-partial-functions (auto_partial), 7
automatically partially-applied function, 306
axis_titles_bottom_left (theme_ggdist), 302

bandwidth, 7, 9, 25, 31
bandwidth_bcv (bandwidth), 9
bandwidth_dpi (bandwidth), 9
bandwidth_dpi(), 9
bandwidth_nrd (bandwidth), 9
bandwidth_nrd0 (bandwidth), 9
bandwidth_nrd0(), 9
bandwidth_SJ (bandwidth), 9
bandwidth_ucv (bandwidth), 9
beeswarm::beeswarm(), 11, 33, 37, 45, 53, 95, 103, 169, 179, 242
bin_dots, 10
bin_dots(), 34
blur, 7, 12
blur_gaussian (blur), 12
blur_gaussian(), 36, 241
blur_interval (blur), 12
blur_interval(), 36, 241
borders(), 38, 47, 56, 63, 68, 72, 78, 84, 90, 97, 105, 151, 162, 171, 181, 193, 203, 213, 224, 231, 237, 244, 252, 258, 265, 275, 286
bounder_cdf, 13, 15, 16
bounder_cdf(), 14, 26, 141
bounder_cooke, 15, 15, 16
bounder_cooke(), 14, 26, 141
bounder_range, 15, 16
bounder_range(), 26, 141
breaks, 6, 7, 16, 28, 150, 160, 191, 201, 212, 222, 264, 274, 284
breaks_FD (breaks), 16
breaks_FD(), 17
breaks_fixed (breaks), 16
breaks_fixed(), 17, 28, 150, 160, 191, 201, 212, 222, 264, 274, 284
breaks_quantiles (breaks), 16
breaks_quantiles(), 17
breaks_Scott (breaks), 16
breaks_Scott(), 17
breaks_Sturges (breaks), 16
breaks_Sturges(), 17, 28, 150, 160, 191, 201, 212, 222, 264, 274, 284
bw.SJ(), 9

cdf(), 23
character, 21, 23, 112, 118–120, 124, 131, 133, 136, 283, 294
continuous_scale(), 133, 138, 300
coord_cartesian(), 133, 138, 294
curve_interval, 19
curve_interval(), 20
cut_cdf_qi, 23
cut_cdf_qi(), 23

data.frame, 20, 115, 118, 124, 305
data.frame(), 119
density_bounded, 24, 30, 32, 125
density_bounded(), 7, 9, 10, 13–16, 125, 141, 142, 149, 151, 160, 161, 190–192, 201, 202, 211, 213, 221–223, 264, 265, 273, 275, 284, 285
density_histogram, 27, 28, 32
density_histogram(), 5–7, 10, 16–18, 149, 150, 160, 161, 190, 191, 201, 202, 211, 212, 221–223, 264, 265, 273, 274, 284, 285
density_unbounded, 27, 30, 30, 125
density_unbounded(), 7, 9, 10, 141, 142, 144, 149, 160, 190, 201, 211, 221, 264, 273, 284
discrete_scale(), 133
dist_beta(), 57, 152, 162, 172, 182, 193, 203, 214, 224, 231, 237, 245, 252, 258, 266, 276, 286
dist_normal(), 57, 152, 153, 162, 163, 172, 173, 182–184, 193, 194, 203–205,

214, 215, 224, 225, 231, 232, 237, 238, 245, 246, 252, 253, 258, 259, 266, 267, 276, 277, 286, 287
dist_truncated(), 119
distributional::dist_wrap(), 118
dlkjcorr_marginal (lkjcorr_marginal), 113
dnorm(), 57, 152, 162, 172, 182, 193, 203, 214, 224, 231, 237, 245, 252, 258, 266, 276, 286
dplyr::group_by(), 20, 124, 125
dstudent_t (student_t), 289

ecdf(), 307, 308
environment, 118, 119
expansion(), 135, 138, 300

facet_title_horizontal (theme_ggdist), 302
facet_title_left_horizontal (theme_ggdist), 302
facet_title_right_horizontal (theme_ggdist), 302
fda::fbplot(), 21
find_dotplot_binwidth, 33
find_dotplot_binwidth(), 12
findInterval(), 29
fortify(), 35, 44, 52, 62, 66, 70, 75, 81, 89, 94, 102, 147, 157, 168, 177, 188, 198, 209, 219, 230, 236, 241, 250, 256, 262, 271, 282
from_broom_names (tidy-format-translators), 304
from_broom_names(), 305
from_ggmcmc_names (tidy-format-translators), 304
from_ggmcmc_names(), 305
function, 8, 17, 23, 25, 28, 31, 36–38, 45, 46, 54, 55, 76, 77, 82, 84, 89, 90, 96, 97, 104, 105, 124, 125, 141, 147–150, 158–161, 169, 170, 179, 180, 188, 190–192, 199–202, 209, 211, 212, 219, 221–223, 230, 236, 241–243, 251, 257, 262–264, 271, 273, 274, 282–285, 300

Geom, 147, 157, 168, 177, 188, 198, 209, 219, 230, 236, 241, 250, 257, 262, 271, 282
geom_blur_dots, 34, 50, 60, 101, 109
geom_blur_dots(), 12, 13, 39, 47, 56, 98, 106, 171, 182, 240, 241, 245, 246, 249
geom_dots, 42, 43, 60, 101, 109, 141, 143
geom_dots(), 7, 10, 34, 39, 47, 56, 97, 106, 168, 171, 173, 176, 182, 244, 298
geom_dotsinterval, 42, 50, 51, 101, 109
geom_dotsinterval(), 4, 12, 34, 39, 42, 43, 47, 50, 51, 56, 88, 93, 97, 101, 106, 109, 134, 143, 167, 171, 177, 178, 182, 184, 186, 244, 294, 296, 298, 299
geom_interval, 61, 74, 80, 92
geom_interval(), 63, 85, 229, 230, 233, 234
geom_line(), 66, 68, 69
geom_lineribbon, 66
geom_lineribbon(), 4, 68, 88, 132, 235, 236, 238, 239, 256, 257, 259, 260
geom_point(), 39, 47, 56, 97, 106, 171, 182, 244
geom_pointinterval, 65, 70, 80, 92
geom_pointinterval(), 69, 72, 85, 249, 250, 253, 255
geom_ribbon(), 66, 68, 69
geom_slab, 65, 74, 75, 92
geom_slab(), 41, 49, 59, 64, 73, 79, 85, 86, 99, 108, 154, 165, 174, 185, 195, 206, 216, 227, 233, 248, 254, 261, 262, 267–269, 278
geom_slabinterval, 80
geom_slabinterval(), 4, 11, 39, 40, 47, 48, 51, 56, 58, 60, 61, 63, 65, 70, 72, 74, 75, 80, 84–86, 88, 91, 97, 99, 105, 107, 111, 127, 132, 134, 137–140, 146, 147, 153, 155–158, 164, 166, 171, 174, 181, 184, 187, 188, 194–199, 205, 207, 209, 215, 217–219, 225, 226, 228, 244, 247, 270, 271, 277, 278, 280, 291, 294, 296–301, 304
geom_spike, 65, 74, 80, 88
geom_spike(), 282, 287, 289, 298
geom_swarm, 42, 50, 60, 93, 109
geom_swarm(), 39, 47, 56, 97, 106, 171, 182, 244
geom_weave, 42, 50, 60, 101, 101
geom_weave(), 39, 47, 56, 97, 106, 171, 182,

244
ggdist (ggdist-package), 4
ggdist-deprecated, 110
ggdist-package, 4
ggplot(), 35, 39, 44, 48, 52, 57, 62, 63, 66, 68, 70, 72, 75, 78, 81, 85, 89, 91, 94, 98, 102, 106, 134, 136, 140, 147, 152, 157, 162, 167, 172, 177, 183, 188, 193, 198, 204, 209, 214, 219, 224, 229, 232, 236, 237, 240, 245, 250, 252, 256, 258, 262, 266, 271, 276, 282, 286, 295
ggplot2, 63, 72
ggplot2::continuous_scale, 138
ggplot2::discrete_scale, 135
ggplot2::Geom, 39, 48, 57, 63, 68, 72, 78, 85, 91, 98, 106
ggplot2::geom_dotplot(), 38, 47, 56, 97, 105, 171, 181, 244
ggplot2::guide_colourbar, 111
ggplot2::position_dodge(), 127
ggplot2::Scale, 38, 46, 55, 77, 84, 89, 97, 105, 134, 136, 140, 148, 159, 170, 180, 190, 200, 211, 221, 243, 263, 273, 282, 295
ggplot2::Stat, 57, 152, 162, 172, 183, 193, 204, 214, 224, 232, 237, 245, 252, 258, 266, 276, 286
ggplot2::theme(), 303
ggplot2::theme_light(), 302
ggplot2::theme_set(), 303
ggplot2::waiver(), 306
graphics::hist(), 28, 150, 160, 191, 201, 211, 212, 222, 264, 273, 274, 284
grDevices::nclass.FD(), 17
grid::grob, 38, 46, 55, 77, 84, 89, 97, 105, 148, 159, 170, 180, 190, 200, 211, 221, 243, 263, 273, 282
grob, 34
grouped_df, 124
Guide, 133, 294
guide_axis(), 297
guide_colorbar2 (sub-geometry-scales), 291
guide_colourbar(), 111
guide_colourbar2 (sub-geometry-scales), 291
guide_rampbar, 111, 121, 132, 134
guide_rampbar(), 112, 133
guides(), 133, 136, 138

hdci (point_interval), 121
hdi, 124
hdi (point_interval), 121
hdi(), 124, 125
hist(), 29

integer, 26, 31, 309

labs(), 111
lambda, 133, 135, 137–139, 294, 297
language, 20, 115, 119, 124, 129
layer position, 67
layer stat, 35, 44, 52, 62, 67, 71, 76, 82, 89, 94, 102
layer(), 36, 44, 53, 62, 67, 71, 76, 82, 89, 94, 103, 147, 158, 168, 178, 188, 199, 209, 219, 230, 236, 241, 250, 257, 262, 271, 282
linetype, 294
list, 283
lkjcorr_marginal, 113
lkjcorr_marginal(), 115, 116
ll (point_interval), 121
logical, 8, 20, 21, 25, 26, 29, 31, 38, 46, 47, 54–56, 63, 67, 72, 77, 84, 90, 96, 97, 104, 105, 119, 124, 138, 142, 149–151, 159–162, 170, 171, 179–181, 191, 192, 201–203, 211–213, 222–224, 231, 236, 237, 243, 244, 251, 252, 257, 258, 264, 265, 273–275, 284–286, 294, 307, 309

make.names(), 119
marginalize_lkjcorr, 115
marginalize_lkjcorr(), 114
matrix, 20
mean, 124
mean_hdci (point_interval), 121
mean_hdi (point_interval), 121
mean_ll (point_interval), 121
mean_qi (point_interval), 121
mean_qi(), 7, 63, 72
mean_ul (point_interval), 121
median, 124
median_hdci (point_interval), 121

median_hdi (point_interval), 121
median_ll (point_interval), 121
median_qi (point_interval), 121
median_qi(), 7, 63, 72
median_ul (point_interval), 121
Mode, 124
Mode (point_interval), 121
Mode(), 125
mode_hdci (point_interval), 121
mode_hdi (point_interval), 121
mode_hdi(), 7, 63, 72
mode_ll (point_interval), 121
mode_qi (point_interval), 121
mode_ul (point_interval), 121

nclass.FD(), 17
nclass.scott(), 17
nclass.Sturges(), 17
NULL, 17, 23, 25, 28, 31, 55, 63, 72, 84, 90, 115, 118, 125, 148, 159, 180, 190, 200, 210, 221, 230, 251, 272, 283, 300, 307, 308
numeric, 5, 9, 10, 13–17, 20, 23, 25, 28, 31, 33, 36, 44, 45, 53, 55, 63, 71, 72, 83, 84, 95, 96, 103, 104, 113, 115, 120, 124, 125, 127, 133, 138, 141, 143, 145, 148–151, 158–161, 168, 178, 180, 181, 189–192, 199–202, 210–213, 221–223, 230, 236, 241, 242, 250, 251, 257, 263–265, 272–275, 283–285, 290, 294, 297, 299, 300, 303, 307–309

ordered, 23

p_ (Pr_), 129
p_(), 129
parse_dist, 117
parse_dist(), 57, 114–116, 119, 152, 162, 172, 182, 193, 203, 214, 224, 231, 232, 237, 245, 252, 258, 266, 276, 286, 290
partial_colour_ramp, 112, 120, 131–134
partial_colour_ramp(), 120, 132, 134
plkjcorr_marginal (lkjcorr_marginal), 113
plot(), 26, 29, 32
pnorm(), 23, 57, 152, 162, 172, 182, 193, 203, 214, 224, 231, 237, 245, 252, 258, 266, 276, 286
point_interval, 121
point_interval(), 7, 22, 63, 66, 68, 72, 150, 161, 167, 177, 180, 181, 192, 202, 212, 223, 230, 231, 235, 236, 251, 256, 257, 274, 306
Position, 36, 44, 52, 62, 71, 76, 82, 89, 94, 103, 147, 157, 168, 177, 188, 199, 209, 219, 230, 236, 241, 250, 257, 262, 271, 282
position_dodge(), 36, 44, 52, 62, 71, 76, 82, 89, 94, 103, 147, 157, 168, 177, 188, 199, 209, 219, 230, 236, 241, 250, 257, 262, 271, 282
position_dodgejust, 127
position_dodgejust(), 36, 44, 52, 62, 71, 76, 82, 89, 94, 103, 127, 147, 157, 168, 177, 188, 199, 209, 219, 230, 236, 241, 250, 257, 262, 271, 282
posterior::mcse_quantile(), 240
posterior::rvar, 20
posterior::rvar(), 20, 39, 47, 56, 57, 97, 98, 106, 152, 153, 162, 163, 171–173, 182–184, 193, 194, 203–205, 214, 215, 224, 225, 231, 232, 237, 238, 244–246, 252, 253, 258, 259, 266, 267, 276, 277, 283, 286, 287
Pr_, 129
Pr_(), 129
print(), 26, 29, 32
pstudent_t (student_t), 289

qi, 124
qi (point_interval), 121
qi(), 124
qlkjcorr_marginal (lkjcorr_marginal), 113
qnorm(), 57, 152, 162, 172, 182, 193, 203, 214, 224, 231, 237, 245, 252, 258, 266, 276, 286
qstudent_t (student_t), 289
quantile(), 308, 309
quasiquotation, 129

r_dist_name (parse_dist), 117
r_dist_name(), 119
ramp_colours, 112, 121, 131, 134
ramp_colours(), 120, 134
resolution(), 143, 144

rlang::eval_tidy(), *125*
rlkjcorr_marginal *(*lkjcorr_marginal*)*, 113
rstudent_t *(*student_t*)*, 289
rvar, *20*

scale_alpha_continuous(), *294*
scale_color_continuous(), *295*
scale_color_discrete(), *294, 295*
scale_color_ramp *(*scale_colour_ramp*)*, 132
scale_color_ramp_continuous *(*scale_colour_ramp*)*, 132
scale_color_ramp_discrete *(*scale_colour_ramp*)*, 132
scale_colour_gradient2(), *139*
scale_colour_gradientn(), *139*
scale_colour_ramp, *112*, *120, 121*, *132*, 132, *136*, *140*, *295*
scale_colour_ramp(), *41*, *49*, *58*, *64*, *73*, *79*, *86*, *91*, *99*, *107*, *154*, *165*, *174*, *185*, *195*, *206*, *216*, *226*, *233*, *247*, *254*, *268*, *278*, *287*
scale_colour_ramp_continuous *(*scale_colour_ramp*)*, 132
scale_colour_ramp_continuous(), *111, 112*, *133*
scale_colour_ramp_discrete *(*scale_colour_ramp*)*, 132
scale_colour_ramp_discrete(), *133*
scale_fill_ramp *(*scale_colour_ramp*)*, 132
scale_fill_ramp(), *41*, *49*, *58*, *64*, *68*, *73*, *79*, *86*, *91*, *99*, *108*, *154*, *165*, *174*, *185*, *195*, *206*, *216*, *226*, *233*, *239*, *247*, *254*, *260*, *268*, *278*, *287*
scale_fill_ramp_continuous *(*scale_colour_ramp*)*, 132
scale_fill_ramp_continuous(), *111, 112*, *133*
scale_fill_ramp_discrete *(*scale_colour_ramp*)*, 132
scale_interval_alpha_continuous *(*sub-geometry-scales*)*, 291
scale_interval_alpha_discrete *(*sub-geometry-scales*)*, 291
scale_interval_color_continuous *(*sub-geometry-scales*)*, 291
scale_interval_color_discrete *(*sub-geometry-scales*)*, 291
scale_interval_colour_continuous *(*sub-geometry-scales*)*, 291
scale_interval_colour_discrete *(*sub-geometry-scales*)*, 291
scale_interval_linetype_continuous *(*sub-geometry-scales*)*, 291
scale_interval_linetype_discrete *(*sub-geometry-scales*)*, 291
scale_interval_size_continuous *(*sub-geometry-scales*)*, 291
scale_interval_size_discrete *(*sub-geometry-scales*)*, 291
scale_point_alpha_continuous *(*sub-geometry-scales*)*, 291
scale_point_alpha_discrete *(*sub-geometry-scales*)*, 291
scale_point_color_continuous *(*sub-geometry-scales*)*, 291
scale_point_color_discrete *(*sub-geometry-scales*)*, 291
scale_point_colour_continuous *(*sub-geometry-scales*)*, 291
scale_point_colour_discrete *(*sub-geometry-scales*)*, 291
scale_point_fill_continuous *(*sub-geometry-scales*)*, 291
scale_point_fill_discrete *(*sub-geometry-scales*)*, 291
scale_point_size_continuous *(*sub-geometry-scales*)*, 291
scale_point_size_continuous(), *55*, *72*, *84*, *148*, *159*, *180*, *190*, *200*, *210*, *221*, *251*, *272*
scale_point_size_discrete *(*sub-geometry-scales*)*, 291
scale_point_size_discrete(), *55*, *72*, *84*, *148*, *159*, *180*, *190*, *200*, *210*, *221*, *251*, *272*
scale_side_mirrored, *134*, 134, *140*, *295*
scale_size_continuous(), *55*, *63*, *71*, *83*, *85*, *148*, *158, 159*, *180*, *189*, *199*, *210*, *221*, *230*, *250, 251*, *272*
scale_slab_alpha_continuous *(*sub-geometry-scales*)*, 291
scale_slab_alpha_discrete *(*sub-geometry-scales*)*, 291
scale_slab_color_continuous *(*sub-geometry-scales*)*, 291

scale_slab_color_discrete
(sub-geometry-scales), 291
scale_slab_colour_continuous
(sub-geometry-scales), 291
scale_slab_colour_discrete
(sub-geometry-scales), 291
scale_slab_fill_continuous
(sub-geometry-scales), 291
scale_slab_fill_discrete
(sub-geometry-scales), 291
scale_slab_linetype_continuous
(sub-geometry-scales), 291
scale_slab_linetype_discrete
(sub-geometry-scales), 291
scale_slab_linewidth_continuous
(sub-geometry-scales), 291
scale_slab_linewidth_discrete
(sub-geometry-scales), 291
scale_slab_shape_continuous
(sub-geometry-scales), 291
scale_slab_shape_discrete
(sub-geometry-scales), 291
scale_slab_size_continuous
(sub-geometry-scales), 291
scale_slab_size_discrete
(sub-geometry-scales), 291
scale_thickness, *134*, *136*, 137, *295*
scale_thickness_identity
(scale_thickness), 137
scale_thickness_shared
(scale_thickness), 137
scale_thickness_shared(), *139*, *298*, *301*, *304*
scales, *42*, *50*, *60*, *65*, *69*, *74*, *79*, *87*, *92*, *100*, *109*, *155*, *166*, *175*, *186*, *196*, *207*, *217*, *228*, *234*, *239*, *249*, *255*, *260*, *269*, *279*, *288*
scales (sub-geometry-scales), 291
scales::censor(), *138*
scales::extended_breaks(), *137*, *297*
scales::new_transform(), *139*
scales::pal_area(), *138*
scales::pal_hue(), *135*
scales::percent_format(), *23*
scales::rescale(), *139*
scales::squish(), *138*
scales::squish_infinite(), *138*
smooth_, *26*, *29*, *32*
smooth_bar (smooth_discrete), 143
smooth_bar(), *7*, *144*
smooth_bounded (smooth_density), 141
smooth_bounded(), *7*
smooth_density, 141, *144, 145*
smooth_discrete, *142*, 143, *145*
smooth_discrete(), *7*, *144*
smooth_none, *142*, *144*, 145
smooth_unbounded (smooth_density), 141
smooth_unbounded(), *7*, *144*
stage(), *152*, *163*, *172*, *183*, *193*, *204*, *214*, *224*, *232*, *238*, *245*, *252*, *258*, *266*, *276*, *288*
stat_ccdfinterval, 146, *166*, *196*, *207*, *217*, *228*, *234*, *255*, *269*, *289*
stat_ccdfinterval(), *147*, *276*, *304*
stat_cdfinterval, *155*, 156, *196*, *207*, *217*, *228*, *234*, *255*, *269*, *289*
stat_cdfinterval(), *157*, *276*
stat_dist_ccdfinterval
(ggdist-deprecated), 110
stat_dist_cdfinterval
(ggdist-deprecated), 110
stat_dist_dots (ggdist-deprecated), 110
stat_dist_dotsinterval
(ggdist-deprecated), 110
stat_dist_eye (ggdist-deprecated), 110
stat_dist_gradientinterval
(ggdist-deprecated), 110
stat_dist_halfeye (ggdist-deprecated), 110
stat_dist_interval (ggdist-deprecated), 110
stat_dist_lineribbon
(ggdist-deprecated), 110
stat_dist_pointinterval
(ggdist-deprecated), 110
stat_dist_slab (ggdist-deprecated), 110
stat_dist_slabinterval
(ggdist-deprecated), 110
stat_dots, 167, *186*, *249*
stat_dots(), *39*, *47, 48*, *50, 51*, *56, 57*, *97, 98*, *106*, *167, 168*, *171*, *182*, *240*, *244, 245*
stat_dotsinterval, *176*, 176, *249*
stat_dotsinterval(), *4*, *39*, *47, 48*, *51*, *56, 57*, *98*, *106*, *129*, *171*, *177*, *182*, *244, 245*

stat_eye, 155, 166, 187, 207, 217, 228, 234, 255, 269, 289
stat_eye(), 125, 188, 276
stat_gradientinterval, 155, 166, 196, 197, 217, 228, 234, 255, 269, 289
stat_gradientinterval(), 77, 83, 147, 158, 189, 198, 200, 210, 220, 263, 272, 276, 294, 304
stat_halfeye, 155, 166, 196, 207, 208, 228, 234, 255, 269, 289
stat_halfeye(), 88, 117, 125, 209, 276
stat_histinterval, 155, 166, 196, 207, 217, 218, 234, 255, 269, 289
stat_histinterval(), 219, 276
stat_interval, 155, 166, 196, 207, 217, 228, 229, 255, 269, 289
stat_interval(), 65, 230, 276
stat_lineribbon, 235, 260
stat_lineribbon(), 4, 68, 69, 235, 236, 239, 259
stat_mcse_dots, 176, 186, 240
stat_mcse_dots(), 12, 13, 39, 48, 57, 98, 106, 171, 182, 241, 245
stat_pointinterval, 155, 166, 196, 207, 217, 228, 234, 249, 269, 289
stat_pointinterval(), 74, 250, 276
stat_ribbon, 239, 256
stat_ribbon(), 256, 257
stat_sample_slabinterval (ggdist-deprecated), 110
stat_slab, 155, 166, 196, 207, 217, 228, 234, 255, 261, 289
stat_slab(), 80, 262, 276
stat_slabinterval, 270
stat_slabinterval(), 4, 23, 26, 29, 32, 51, 60, 85, 88, 110, 114, 116, 117, 119, 129, 130, 140, 146, 155, 156, 166, 167, 177, 187, 196, 197, 207, 208, 217, 218, 228, 229, 234, 235, 249, 255, 256, 261, 269, 271, 281, 289, 290
stat_spike, 155, 166, 196, 207, 217, 228, 234, 255, 269, 281
stat_spike(), 92, 282
stat_summary(), 125
StatDistSlabinterval (ggdist-deprecated), 110
stats::bw.SJ, 9
stats::density(), 25, 26, 29–31, 143, 149, 160, 190, 201, 211, 221, 264, 273, 284
StatSampleSlabinterval (ggdist-deprecated), 110
stepfun(), 308
string, 8, 10, 11, 17, 20, 25, 28, 31, 33, 36–38, 44–46, 52–55, 62, 67, 71, 76, 77, 82–84, 89, 90, 94–97, 103–105, 112, 116, 118, 119, 125, 133, 135, 141, 143, 147–151, 157–161, 168–170, 177–181, 188–192, 198–202, 209–213, 219–223, 230, 231, 236, 237, 241–243, 250, 251, 257, 262–265, 271–275, 282–285, 294, 297
student_t, 289
sub-geometry-scales, 55, 63, 72, 84, 148, 159, 180, 190, 200, 210, 221, 230, 251, 272, 291
subguide_axis, 296, 299
subguide_axis(), 38, 46, 55, 77, 84, 89, 97, 105, 148, 159, 170, 180, 190, 200, 211, 221, 243, 263, 273, 282
subguide_count (subguide_axis), 296
subguide_count(), 298
subguide_dots, 38, 47, 56, 77, 84, 89, 97, 105, 149, 159, 170, 180, 190, 200, 211, 221, 243, 263, 273, 283
subguide_dots (subguide_axis), 296
subguide_dots(), 298
subguide_inside (subguide_axis), 296
subguide_inside(), 298
subguide_integer (subguide_axis), 296
subguide_integer(), 298
subguide_none, 298, 299
subguide_none(), 38, 46, 55, 77, 84, 89, 97, 105, 148, 159, 170, 180, 190, 200, 211, 221, 243, 263, 273, 282, 298
subguide_outside (subguide_axis), 296
subguide_outside(), 298
subguide_slab, 38, 47, 56, 77, 84, 89, 97, 105, 149, 159, 170, 180, 190, 200, 211, 221, 243, 263, 273, 283
subguide_slab (subguide_axis), 296
subguide_slab(), 298
subguide_spike, 38, 47, 56, 77, 84, 89, 97, 105, 149, 159, 170, 180, 190, 200,

211, *221*, *243*, *263*, *273*, *283*
subguide_spike *(*subguide_axis*)*, 296
subguide_spike(), *298*
subscale_identity, 299, *301*
subscale_identity(), *76*, *82*, *90*, *147*, *158*, *188*, *199*, *209*, *220*, *262*, *271*, *283*
subscale_thickness, *76*, *82*, *90*, *147*, *158*, *188*, *199*, *209*, *220*, *262*, *271*, *283*, *300*, 300
subscale_thickness(), *76*, *82*, *90*, *140*, *147*, *158*, *188*, *199*, *209*, *220*, *262*, *271*, *283*, *298*, *304*

theme, *111*, *297*
theme_ggdist, 302
theme_ggdist(), *302*
theme_tidybayes *(*theme_ggdist*)*, 302
theme_tidybayes(), *302*
thickness, *76*, *82*, *90*, *140*, *147*, *158*, *188*, *199*, *209*, *220*, *262*, *271*, *283*, *298–301*, 303, *304*
thickness(), *139*
tidy-format-translators, 304
tidyselect, *20*
to_broom_names *(*tidy-format-translators*)*, 304
to_broom_names(), *305*
to_ggmcmc_names *(*tidy-format-translators*)*, 304
to_ggmcmc_names(), *305*
transformation object, *137*, *297*

ul *(*point_interval*)*, 121
uniroot, *9*
unit, *36*, *44*, *53*, *95*, *103*, *104*, *168*, *178*, *241*
unit(), *36*, *38*, *44*, *46*, *53*, *54*, *95*, *96*, *104*, *168*, *170*, *178*, *179*, *241*, *243*

vctrs::rcrd, *120*, *304*
vctrs::vec_cast(), *303*

waived arguments, *5*, *9*, *12*, *13*, *15*, *16*, *25*, *28*, *30*, *141*, *143*, *145*, *296*, *299*, *300*
waiver, *7*, *149*, *150*, *160*, *161*, *191*, *201*, *202*, *211*, *212*, *222*, *223*, *264*, *265*, *273*, *274*, *284*, *285*, 306
waiver(), *7*, *8*, *111*, *149*, *150*, *160*, *161*, *191*, *201*, *202*, *211*, *212*, *222*, *223*, *264*, *265*, *273*, *274*, *284*, *285*, *306*
weighted_ecdf, 307
weighted_ecdf(), *26*, *29*, *32*, *308*, *310*
weighted_quantile, 308
weighted_quantile(), *308*
weighted_quantile_fun *(*weighted_quantile*)*, 308