Package ‘ggforce’

August 20, 2019

Type Package
Title Accelerating 'ggplot2'
Version 0.3.1
Maintainer Thomas Lin Pedersen <thomasp85@gmail.com>
Description The aim of 'ggplot2' is to aid in visual data investigations. This focus has led to a lack of facilities for composing specialised plots. 'ggforce' aims to be a collection of mainly new stats and geoms that fills this gap. All additional functionality is aimed to come through the official extension system so using 'ggforce' should be a stable experience.
URL https://ggforce.data-imaginist.com
BugReports https://github.com/thomasp85/ggforce/issues
License MIT + file LICENSE
LazyData TRUE
Encoding UTF-8
Depends ggplot2 (>= 3.0.0), R (>= 3.3.0)
Imports Rcpp (>= 0.12.2), grid, scales, MASS, tweenr (>= 0.1.5), gtable, rlang, polyclip, stats, grDevices, tidyselect, withr, utils
LinkingTo Rcpp, RcppEigen
RoxygenNote 6.1.1
Suggests sessioninfo, concaveman, deldir, reshape2, units (>= 0.4-6)
Collate 'RcppExports.R' 'aaa.R' 'shape.R' 'arc_bar.R' 'arc.R'
 'autodensity.R' 'autohistogram.R' 'autopoint.R' 'bezier.R'
 'bspline.R' 'bspline_closed.R' 'circle.R' 'diagonal.R'
 'diagonal_wide.R' 'ellipse.R' 'facet_grid_paginate.R'
 'facet_matrix.R' 'facet_row.R' 'facet_stereo.R'
 'facet_wrap_paginate.R' 'facet_zoom.R' 'ggforce_package.R'
 'ggproto-classes.R' 'interpolate.R' 'link.R' 'mark_circle.R'
 'mark_ellipse.R' 'mark_hull.R' 'mark_label.R' 'mark_rect.R'
 'parallel_sets.R' 'position-jitternormal.R' 'position_auto.R'
 'position_floatstack.R' 'regon.R' 'scale-depth.R'

R topics documented:

'scale-unit.R' 'sina.R' 'spiro.R' 'themes.R' 'trans.R' 'trans_linear.R' 'utilities.R' 'voronoi.R'

**NeedsCompilation** yes

**Author** Thomas Lin Pedersen [cre, aut]

(<https://orcid.org/0000-0002-5147-4711>), RStudio [cph]

**Repository** CRAN

**Date/Publication** 2019-08-20 09:50:05 UTC

R topics documented:

- ggforce-package ........................................................................ 3
- facet_grid_paginate .................................................................. 4
- facet_matrix ........................................................................... 6
- facet_row .............................................................................. 8
- facet_stereo .......................................................................... 9
- facet_wrap_paginate .............................................................. 10
- facet_zoom .......................................................................... 12
- gather_set_data ...................................................................... 13
- GeomShape .......................................................................... 14
- geom_arc ............................................................................. 14
- geom_arc_bar ....................................................................... 17
- geom_autodensity ................................................................. 20
- geom_autopoint .................................................................... 22
- geom_bezier ....................................................................... 23
- geom_bspline ...................................................................... 26
- geom_bspline_closed ........................................................... 29
- geom_circle ....................................................................... 32
- geom_diagonal ..................................................................... 34
- geom_diagonal_wide ............................................................. 37
- geom_ellipse ...................................................................... 39
- geom_link .......................................................................... 41
- geom_mark_circle ................................................................ 44
- geom_mark_ellipse ............................................................... 47
- geom_mark_hull ................................................................. 51
- geom_mark_rect ................................................................... 55
- geom_parallel_sets .............................................................. 58
- geom_region ...................................................................... 61
- geom_shape ...................................................................... 63
- geom_sina .......................................................................... 65
- geom_spro .......................................................................... 69
- geom_voronoi .................................................................... 71
- linear_trans ........................................................................ 75
- n_pages ............................................................................. 76
- position_auto ...................................................................... 76
- position_jitternormal .......................................................... 77
**ggforce-package**

**Description**

The aim of `ggplot2` is to aid in visual data investigations. This focus has led to a lack of facilities for composing specialised plots. `ggforce` aims to be a collection of mainly new stats and geoms that fills this gap. All additional functionality is aimed to come through the official extension system so using `ggforce` should be a stable experience.

**Author(s)**

**Maintainer:** Thomas Lin Pedersen <thomasp85@gmail.com> (0000-0002-5147-4711)

Other contributors:
- RStudio [copyright holder]

**See Also**

Useful links:
- [https://ggforce.data-imaginist.com](https://ggforce.data-imaginist.com)

**Examples**

```r
rocketData <- data.frame(
  x = c(1, 1, 2, 2),
  y = c(1, 2, 2, 3)
)
rocketData <- do.call(rbind, lapply(seq_len(500) - 1, function(i) {
  rocketData$y <- rocketData$y - c(0, i / 500)
  rocketData$group <- 1 + 1
  rocketData
})
)
rocketData2 <- data.frame(
  x = c(2, 2.25, 2),
  y = c(2, 2.5, 3)
)
rocketData2 <- do.call(rbind, lapply(seq_len(500) - 1, function(i) {
})
)```

rocketData2$group <- i + 1 + 500
rocketData2

```r
ggplot() + geom_line(aes(
  x = 2, y = 2, xend = 3, yend = 3, alpha = ..index..,
  size = ..index..
), colour = 'goldenrod', n = 500) +
  geom_bezier(aes(x = x, y = y, group = group, colour = ..index..),
  data = rocketData
) +
  geom_bezier(aes(x = y, y = x, group = group, colour = ..index..),
  data = rocketData
) +
  geom_bezier(aes(x = x, y = y, group = group, colour = 1),
  data = rocketData2
) +
  geom_bezier(aes(x = y, y = x, group = group, colour = 1),
  data = rocketData2
) +
  geom_text(aes(x = 1.65, y = 1.65, label = 'ggplot2', angle = 45),
  colour = 'white', size = 15)
) +
  coord_fixed() +
  scale_x_reverse() +
  scale_y_reverse() +
  scale_alpha(range = c(1, 0), guide = 'none') +
  scale_size_continuous(
    range = c(20, 0.1), trans = 'exp',
    guide = 'none'
  ) +
  scale_color_continuous(guide = 'none') +
  xlab('') + ylab('') +
  ggtitle('ggforce: Accelerating ggplot2') +
  theme(plot.title = element_text(size = 20))
```

### facet_grid_paginate

**Split facet_grid over multiple plots**

**Description**

This extension to `ggplot2::facet_grid()` will allow you to split a faceted plot over multiple pages. You define a number of rows and columns per page as well as the page number to plot, and the function will automatically only plot the correct panels. Usually this will be put in a loop to render all pages one by one.

**Usage**

```r
facet_grid_paginate(facets, margins = FALSE, scales = "fixed",
  space = "fixed", shrink = TRUE, labeller = "label_value",
```
as.table = TRUE, switch = NULL, drop = TRUE, ncol = NULL, nrow = NULL, page = 1, byrow = TRUE)

Arguments

facets This argument is soft-deprecated, please use rows and cols instead.
margins Either a logical value or a character vector. Margins are additional facets which contain all the data for each of the possible values of the faceting variables. If FALSE, no additional facets are included (the default). If TRUE, margins are included for all faceting variables. If specified as a character vector, it is the names of variables for which margins are to be created.
scales Are scales shared across all facets (the default, "fixed"), or do they vary across rows ("free_x"), columns ("free_y"), or both rows and columns ("free")?

space If "fixed", the default, all panels have the same size. If "free_y" their height will be proportional to the length of the y scale; if "free_x" their width will be proportional to the length of the x scale; or if "free" both height and width will vary. This setting has no effect unless the appropriate scales also vary.

shrink If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.

labeller A function that takes one data frame of labels and returns a list or data frame of character vectors. Each input column corresponds to one factor. Thus there will be more than one with formulae of the type ~cyl + am. Each output column gets displayed as one separate line in the strip label. This function should inherit from the "labeller" S3 class for compatibility with labeller(). See label_value() for more details and pointers to other options.

as.table If TRUE, the default, the facets are laid out like a table with highest values at the bottom-right. If FALSE, the facets are laid out like a plot with the highest value at the top-right.

switch By default, the labels are displayed on the top and right of the plot. If "x", the top labels will be displayed to the bottom. If "y", the right-hand side labels will be displayed to the left. Can also be set to "both".

drop If TRUE, the default, all factor levels not used in the data will automatically be dropped. If FALSE, all factor levels will be shown, regardless of whether or not they appear in the data.

ncol Number of columns per page

nrow Number of rows per page

page The page to draw

byrow Should the pages be created row-wise or column wise

Note

If either ncol or nrow is NULL this function will fall back to the standard facet_grid functionality.

See Also

n_pages() to compute the total number of pages in a paginated faceted plot

Other ggforce facets: facet_stereo, facet_wrap_paginate, facet_zoom
## facet_matrix

**Facet by different data columns**

The `facet_matrix()` facet allows you to put different data columns into different rows and columns in a grid of panels. If the same data columns are present in both the rows and the columns of the grid, and used together with `ggplot2::geom_point()` it is also known as a scatterplot matrix, and if other geoms are used it is sometimes referred to as a pairs plot. `facet_matrix` is so flexible that these types are simply a subset of its capabilities, as any combination of data columns can be plotted against each other using any type of geom. Layers should use the `panel_x` and `panel_y` placeholders to map aesthetics to, in order to access the row and column data.

### Usage

```r
facet_matrix(rows, cols = rows, shrink = TRUE, switch = NULL, flip.rows = FALSE, alternate.axes = FALSE, layer.lower = NULL, layer.diag = NULL, layer.upper = NULL, layer.continuous = NULL, layer.discrete = NULL, layer.mixed = NULL, grid.y.diag = TRUE)
```

### Arguments

- **rows, cols**  A specification of the data columns to put in the rows and columns of the facet grid. They are specified using the `ggplot2::vars()` function wherein you can use standard tidyselect syntax as known from e.g. `dplyr::select()`. These data values will be made available to the different layers through the `panel_x` and `panel_y` variables.

- **shrink**  If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.

- **switch**  By default, the labels are displayed on the top and right of the plot. If "x", the top labels will be displayed to the bottom. If "y", the right-hand side labels will be displayed to the left. Can also be set to "both".

- **flip.rows**  Should the order of the rows be reversed so that, if the rows and columns are equal, the diagonal goes from bottom-left to top-right instead of top-left to bottom-right.

- **alternate.axes**  Should axes be drawn at alternating positions.

- **layer.lower, layer.diag, layer.upper**  Specification for where each layer should appear. The default (NULL) will allow any layer that has not been specified directly to appear at that position. Putting
facet_matrix

e.g. layer.diag = 2 will make the second layer appear on the diagonal as well as remove that layer from any position that has NULL. Using TRUE will put all layers at that position, and using FALSE will conversely remove all layers. These settings will only have an effect if the grid is symmetric.

layer.continuous, layer.discrete, layer.mixed

As above, but instead of referencing panel positions it references the combination of position scales in the panel. Continuous panels have both a continuous x and y axis, discrete panels have both a discrete x and y axis, and mixed panels have one of each. Unlike the position based specifications above these also have an effect in non-symmetric grids.

grid.y.diag

Should the y grid be removed from the diagonal? In certain situations the diagonal are used to plot the distribution of the column data and will thus not use the y-scale. Removing the y gridlines can indicate this.

Note

Due to the special nature of this faceting it slightly breaks the ggplot2 API, in that any positional scale settings are ignored. This is because each row and column in the grid will potentially have very different scale types and it is not currently possible to have multiple different scale specifications in the same plot object.

See Also

geomautopoint, geomautohistogram, geomautodensity, and positionauto for geoms and positions that adapts to different positional scale types

Examples

# Standard use:
ggplot(mpg) + geom_point(aes(x = .panel_x, y = .panel_y)) + facet_matrix(vars(displ, cty, hwy))

# Switch the diagonal, alternate the axes and style strips as axis labels
ggplot(mpg) + geom_point(aes(x = .panel_x, y = .panel_y)) + facet_matrix(vars(displ, cty, hwy), flip.rows = TRUE, alternate.axes = TRUE, switch = 'both') + theme(strip.background = element_blank(), strip.placement = 'outside', strip.text = element_text(size = 12))

# Mix discrete and continuous columns. Use geomautopoint for scale-based jitter
ggplot(mpg) + geomautopoint() + facet_matrix(vars(drv:fl))

# Have a special diagonal layer
ggplot(mpg) + geomautopoint() + geomautodensity() +
facet_matrix(vars(drv:fl), layer.diag = 2)

# Show continuous panels in upper triangle as contours and rest as binned
ggplot(mpg) +
  geom_autopoint() +
  geom_autodensity() +
  geom_density2d(aes(x = .panel_x, y = .panel_y)) +
  geom_bin2d(aes(x = .panel_x, y = .panel_y)) +
  facet_matrix(vars(drv:fl), layer.lower = 1, layer.diag = 2,
               layer.continuous = -4, layer.discrete = -3, layer.mixed = -3)

# Make asymmetric grid
ggplot(mpg) +
  geom_boxplot(aes(x = .panel_x, y = .panel_y, group = .panel_x)) +
  facet_matrix(rows = vars(cty, hwy), cols = vars(drv, fl))

---

**facet_row**  
*One-dimensional facets*

**Description**

These facets are one-dimensional versions of `ggplot2::facet_wrap()`, arranging the panels in either a single row or a single column. This restriction makes it possible to support a space argument as seen in `ggplot2::facet_grid()` which, if set to "free" will allow the panels to be sized based on the relative range of their scales. Another way of thinking about them are one-dimensional versions of `ggplot2::facet_grid()` (ie. `~ {var}` or `{var} ~ .`), but with the ability to position the strip at either side of the panel. However you look at it it is the best of both world if you just need one dimension.

**Usage**

```r
facet_row(facets, scales = "fixed", space = "fixed", shrink = TRUE, 
          labeller = "label_value", drop = TRUE, strip.position = "top")
```

```r
facet_col(facets, scales = "fixed", space = "fixed", shrink = TRUE, 
          labeller = "label_value", drop = TRUE, strip.position = "top")
```

**Arguments**

- **facets**
  A set of variables or expressions quoted by `vars()` and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to `labeller`).
  For compatibility with the classic interface, can also be a formula or character vector. Use either a one sided formula, `~a + b`, or a character vector, `c("a","b")`.
scales  Should scales be fixed ("fixed", the default), free ("free"), or free in one dimension ("free_x", "free_y")?
space   Should the size of the panels be fixed or relative to the range of the respective position scales
shrink  If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.
labeller A function that takes one data frame of labels and returns a list or data frame of character vectors. Each input column corresponds to one factor. Thus there will be more than one with formulae of the type ~cyl + am. Each output column gets displayed as one separate line in the strip label. This function should inherit from the "labeller" S3 class for compatibility with labeller(). See label_value() for more details and pointers to other options.
drop    If TRUE, the default, all factor levels not used in the data will automatically be dropped. If FALSE, all factor levels will be shown, regardless of whether or not they appear in the data.
strip.position By default, the labels are displayed on the top of the plot. Using strip.position it is possible to place the labels on either of the four sides by setting strip.position = c("top","bottom","left","right")

Examples

# Standard use
ggplot(mtcars) +
  geom_point(aes(disp, mpg)) +
  facet_col(~gear)
# It retains the ability to have unique scales for each panel
ggplot(mtcars) +
  geom_point(aes(disp, mpg)) +
  facet_col(~gear, scales = 'free')
# But can have free sizing along the stacking dimension
ggplot(mtcars) +
  geom_point(aes(disp, mpg)) +
  facet_col(~gear, scales = 'free', space = 'free')
# And you can position the strip where-ever you like
ggplot(mtcars) +
  geom_point(aes(disp, mpg)) +
  facet_col(~gear, scales = 'free', space = 'free', strip.position = 'bottom')
Description

This, arguably pretty useless function, lets you create plots with a sense of depth by creating two slightly different versions of the plot that corresponds to how the eyes would see it if the plot was 3 dimensional. To experience the effect look at the plots through 3D hardware such as Google Cardboard or by relaxing the eyes and focusing into the distance. The depth of a point is calculated for layers having a depth aesthetic supplied. The scaling of the depth can be controlled with `scale_depth()` as you would control any aesthetic. Negative values will result in features placed behind the paper plane, while positive values will result in features hovering in front of the paper. While features within each layer is sorted so those closest to you are plotted on top of those more distant, this cannot be done between layers. Thus, layers are always plotted on top of each others, even if the features in one layer lies behind features in a layer behind it. The depth experience is inaccurate and should not be used for conveying important data. Regard this more as a party-trick...

Usage

```r
facet_stereo(IPD = 63.5, panel.size = 200, shrink = TRUE)
```

Arguments

- **IPD**: The interpupillary distance (in mm) used for calculating point displacement. The default value is an average of both genders.
- **panel.size**: The final plot size in mm. As IPD this is used to calculate point displacement. Don’t take this value too literal but experiment until you get a nice effect. Lower values gives higher displacement and thus require the plots to be observed from a closer distance.
- **shrink**: If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.

See Also

Other ggforce facets: `facet_grid_paginate`, `facet_wrap_paginate`, `facet_zoom`

Examples

```r
# You'll have to accept a warning about depth being an unknown aesthetic
ggplot(mtcars) +
  geom_point(aes(mpg, disp, depth = cyl)) +
  facet_stereo()
```

Description

This extension to `ggplot2::facet_wrap()` will allow you to split a faceted plot over multiple pages. You define a number of rows and columns per page as well as the page number to plot, and the function will automatically only plot the correct panels. Usually this will be put in a loop to render all pages one by one.
Usage

```r
facet_wrap_paginate(facets, nrow = NULL, ncol = NULL,
                   scales = "fixed", shrink = TRUE, labeller = "label_value",
                   as.table = TRUE, switch = NULL, drop = TRUE, dir = "h",
                   strip.position = "top", page = 1)
```

Arguments

- **facets**: A set of variables or expressions quoted by `vars()` and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to `labeller`). For compatibility with the classic interface, can also be a formula or character vector. Use either a one sided formula, `~a + b`, or a character vector, `c("a","b")`.

- **nrow, ncol**: Number of rows and columns.

- **scales**: Should scales be fixed ("fixed", the default), free ("free"), or free in one dimension ("free_x","free_y")?

- **shrink**: If `TRUE`, will shrink scales to fit output of statistics, not raw data. If `FALSE`, will be range of raw data before statistical summary.

- **labeller**: A function that takes one data frame of labels and returns a list or data frame of character vectors. Each input column corresponds to one factor. Thus there will be more than one with formulae of the type `~cyl + am`. Each output column gets displayed as one separate line in the strip label. This function should inherit from the "labeller" S3 class for compatibility with `labeller()`. See `label_value()` for more details and pointers to other options.

- **as.table**: If `TRUE`, the default, the facets are laid out like a table with highest values at the bottom-right. If `FALSE`, the facets are laid out like a plot with the highest value at the top-right.

- **switch**: By default, the labels are displayed on the top and right of the plot. If "x", the top labels will be displayed to the bottom. If "y", the right-hand side labels will be displayed to the left. Can also be set to "both".

- **drop**: If `TRUE`, the default, all factor levels not used in the data will automatically be dropped. If `FALSE`, all factor levels will be shown, regardless of whether or not they appear in the data.

- **dir**: Direction: either "h" for horizontal, the default, or "v", for vertical.

- **strip.position**: By default, the labels are displayed on the top of the plot. Using `strip.position` it is possible to place the labels on either of the four sides by setting `strip.position = c("top","bottom","left","right")`

- **page**: The page to draw

Note

If either `ncol` or `nrow` is `NULL` this function will fall back to the standard `facet_wrap` functionality.
See Also

n_pages() to compute the total number of pages in a paginated faceted plot
Other ggforce facets: facet_grid_paginate, facet_stereo, facet_zoom

Examples

ggplot(diamonds) +
  geom_point(aes(carat, price), alpha = 0.1) +
  facet_wrap_paginate(~ cut:clarity, ncol = 3, nrow = 3, page = 4)

Facet data for zoom with context

Description

This facetting provides the means to zoom in on a subset of the data, while keeping the view of the
full dataset as a separate panel. The zoomed-in area will be indicated on the full dataset panel for
reference. It is possible to zoom in on both the x and y axis at the same time. If this is done it is
possible to both get each zoom separately and combined or just combined.

Usage

facet_zoom(x, y, xy, zoom.data, xlim = NULL, ylim = NULL,
split = FALSE, horizontal = TRUE, zoom.size = 2,
show.area = TRUE, shrink = TRUE)

Arguments

x, y, xy An expression evaluating to a logical vector that determines the subset of data
to zoom in on
zoom.data An expression evaluating to a logical vector. If TRUE the data only shows in the
zoom panels. If FALSE the data only show in the context panel. If NA the data
will show in all panels.
xlim, ylim Specific zoom ranges for each axis. If present they will override x, y, and/or xy.
split If both x and y is given, should each axis zoom be shown separately as well?
Defaults to FALSE
horizontal If both x and y is given and split = FALSE How should the zoom panel be
positioned relative to the full data panel? Defaults to TRUE
zoom.size Sets the relative size of the zoom panel to the full data panel. The default (2)
makes the zoom panel twice the size of the full data panel.
show.area Should the zoom area be drawn below the data points on the full data panel?
Defaults to TRUE.
shrink If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will
be range of raw data before statistical summary.
**gather_set_data**

*Tidy data for use with geom_parallel_sets*

**Description**

This helper function makes it easy to change tidy data into a tidy(er) format that can be used by geom_parallel_sets.

**Usage**

`gather_set_data(data, x, id_name = "id")`
Arguments

data A tidy dataframe with some categorical columns
x The columns to use for axes in the parallel sets diagram
id_name The name of the column that will contain the original index of the row.

Value

A data.frame

Examples

data <- reshape2::melt(Titanic)
head(gather_set_data(data, 1:4))

Description

ggforce makes heavy use of the ggproto class system to extend the functionality of ggplot2. In
general the actual classes should be of little interest to users as the standard ggplot2 api of using
geom_* and stat_* functions for building up the plot is encouraged.

geom_arc

Arrows based on radius and radians

Description

This set of stats and geoms makes it possible to draw circle segments based on a center point, a ra-
dius and a start and end angle (in radians). These functions are intended for cartesian coordinate sys-
tems and makes it possible to create circular plot types without using the ggplot2::coord_polar() coordinate system.

Usage

stat_arc(mapping = NULL, data = NULL, geom = "arc",
  position = "identity", na.rm = FALSE, show.legend = NA, n = 360,
inherit.aes = TRUE, ...)

geom_arc(mapping = NULL, data = NULL, stat = "arc",
  position = "identity", n = 360, arrow = NULL, lineend = "butt",
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)

stat_arc2(mapping = NULL, data = NULL, geom = "path_interpolate",
  position = "identity", na.rm = FALSE, show.legend = NA, n = 360,
geom_arc

  inherit.aes = TRUE, ...)

geom_arc2(mapping = NULL, data = NULL, stat = "arc2",
  position = "identity", n = 360, arrow = NULL, lineend = "butt",
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)

stat_arc0(mapping = NULL, data = NULL, geom = "arc0",
  position = "identity", na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE, ...)

geom_arc0(mapping = NULL, data = NULL, stat = "arc0",
  position = "identity", ncp = 5, arrow = NULL, lineend = "butt",
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)

Arguments

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

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

geom  
The geometric object to use display the data

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

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

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

n  
the smoothness of the arc. Sets the number of points to use if the arc would cover a full circle

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().

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

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

arrow Arrow specification, as created by `grid::arrow()`.
lineend Line end style (round, butt, square).
ncp the number of control points used to draw the arc with curveGrob. Determines how well the arc approximates a circle section

Details

An arc is a segment of a line describing a circle. It is the fundamental visual element in donut charts where the length of the segment (and conversely the angular span of the segment) describes the proportion of an entity.

Aesthetics

`geom_arc` understand the following aesthetics (required aesthetics are in bold):

- x0
- y0
- r
- start
- end
- color
- size
- linetype
- alpha
- lineend

Computed variables

x, y The start coordinates for the segment
xend, yend The end coordinates for the segment
curvature The curvature of the curveGrob to match a circle

See Also

`geom_arc_bar()` for drawing arcs with fill

Examples

```r
# Lets make some data
arcs <- data.frame(
  start = seq(0, 2 * pi, length.out = 11)[-11],
  end = seq(0, 2 * pi, length.out = 11)[-1],
  r = rep(1:2, 5)
)

# Behold the arcs
ggplot(arcs) +
```


**geom_arc_bar**

`geom_arc(aes(x0 = 0, y0 = 0, r = r, start = start, end = end, linetype = factor(r)))`

# Use the calculated index to map values to position on the arc
`ggplot(arcs) + geom_arc(aes(x0 = 0, y0 = 0, r = r, start = start, end = end, size = stat(index)), lineend = 'round') + scale_radius() # linear size scale`

# The 0 version maps directly to curveGrob instead of calculating the points # itself
`ggplot(arcs) + geom_arc0(aes(x0 = 0, y0 = 0, r = r, start = start, end = end, linetype = factor(r)))`

# The 2 version allows interpolation of aesthetics between the start and end # points
`arcs2 <- data.frame(
  angle = c(arcs$start, arcs$end),
  r = rep(arcs$r, 2),
  group = rep(1:10, 2),
  colour = sample(letters[1:5], 20, TRUE)
)

ggplot(arcs2) + geom_arc2(aes(x0 = 0, y0 = 0, r = r, end = angle, group = group, colour = colour), size = 2)

---

**geom_arc_bar**  
**Arches and wedges as polygons**

**Description**

This set of stats and geoms makes it possible to draw arcs and wedges as known from pie and donut charts as well as more specialized plottypes such as sunburst plots.

**Usage**

`stat_arc_bar(mapping = NULL, data = NULL, geom = "arc_bar", position = "identity", n = 360, na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)`

`stat_pie(mapping = NULL, data = NULL, geom = "arc_bar", position = "identity", n = 360, sep = 0, na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)`

`geom_arc_bar(mapping = NULL, data = NULL, stat = "arc_bar", position = "identity", n = 360, expand = 0, radius = 0, na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)`
Arguments

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

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

gem The geometric object to use display the data

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

n The number of points used to draw a full circle. The number of points on each arc will then be calculated as n / span-of-arc

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

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

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

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

sep The separation between arcs in pie/donut charts

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

expand A numeric or unit vector of length one, specifying the expansion amount. Negative values will result in contraction instead. If the value is given as a numeric it will be understood as a proportion of the plot area width.

radius As expand but specifying the corner radius.

Details

An arc bar is the thick version of an arc; that is, a circle segment drawn as a polygon in the same way as a rectangle is a thick version of a line. A wedge is a special case of an arc where the inner radius is 0. As opposed to applying `coord_polar` to a stacked bar chart, these layers are drawn in cartesian space, which allows for transformations not possible with the native ggplot2 approach. Most notable of these are the option to explode arcs and wedges away from their center point, thus detaching it from the main pie/donut.
Aesthetics

`geom_arc_bar` understand the following aesthetics (required aesthetics are in bold):

- `x0`
- `y0`
- `r0`
- `r`
- `start` - when using `stat_arc_bar`
- `end` - when using `stat_arc_bar`
- `amount` - when using `stat_pie`
- `explode`
- `color`
- `fill`
- `size`
- `linetype`
- `alpha`

Computed variables

- `x`, `y` x and y coordinates for the polygon
- `x`, `y` The start coordinates for the segment

See Also

- `geom_arc()` for drawing arcs as lines

Examples

```r
# If you know the angle spans to plot it is easy
arcs <- data.frame(
  start = seq(0, 2 * pi, length.out = 11)[-11],
  end = seq(0, 2 * pi, length.out = 11)[-1],
  r = rep(1:2, 5)
)

# Behold the arcs
ggplot(arcs) +
  geom_arc_bar(aes(x0 = 0, y0 = 0, r0 = r - 1, r = r, start = start, end = end, fill = r))

# geom_arc_bar uses geom_shape to draw the arcs, so you have all the possibilities of that as well, e.g. rounding of corners
ggplot(arcs) +
  geom_arc_bar(aes(x0 = 0, y0 = 0, r0 = r - 1, r = r, start = start, end = end, fill = r), radius = unit(4, 'mm'))
```
# If you got values for a pie chart, use stat_pie
states <- c('eaten', "eaten but said you didn't", 'cat took it', 'for tonight', 'will decompose slowly')
pie <- data.frame(
  state = factor(rep(states, 2), levels = states),
  type = rep(c('Pie', 'Donut'), each = 5),
  r0 = rep(c(0, 0.8), each = 5),
  focus = rep(c(0.2, 0, 0, 0, 0), 2),
  amount = c(4, 3, 1, 1.5, 6, 6, 1, 2, 3, 2),
  stringsAsFactors = FALSE)

data.frame(
  state = factor(rep(states, 2), levels = states),
  type = rep(c('Pie', 'Donut'), each = 5),
  r0 = rep(c(0, 0.8), each = 5),
  focus = rep(c(0.2, 0, 0, 0, 0), 2),
  amount = c(4, 3, 1, 1.5, 6, 6, 1, 2, 3, 2),
  stringsAsFactors = FALSE)

# Look at the cakes
geom_autodensity()

## geom_autodensity

A distribution geoms that fills the panel and works with discrete and continuous data

### Description

These versions of the histogram and density geoms have been designed specifically for diagonal plotting with `facet_matrix()`. They differ from `ggplot2::geom_histogram()` and `ggplot2::geom_density()` in that they defaults to mapping x and y to `.panel_x` and `.panel_y` respectively, they ignore the y scale of the panel and fills it out, and they work for both continuous and discrete x scales.

### Usage

```r
geom_autodensity(mapping = NULL, data = NULL, stat = "autodensity",
  position = "floatstack", ..., bw = "nrd0", adjust = 1,
  kernel = "gaussian", n = 512, trim = FALSE, na.rm = FALSE,
  show.legend = NA, inherit.aes = TRUE)
```

```r
geom_autohistogram(mapping = NULL, data = NULL, stat = "autobin",
  position = "floatstack", ..., bins = NULL, na.rm = FALSE,
  show.legend = NA, inherit.aes = TRUE)
```
Arguments

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

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

stat Use to override the default connection between `geom_density` and `stat_density`.

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

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

bw The smoothing bandwidth to be used. If numeric, the standard deviation of the smoothing kernel. If character, a rule to choose the bandwidth, as listed in `stats::bw.nrd()`.

adjust A multiplicative bandwidth adjustment. This makes it possible to adjust the bandwidth while still using the a bandwidth estimator. For example, `adjust = 1/2` means use half of the default bandwidth.

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

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

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

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

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

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

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

`facet_matrix` for creating matrix grids

Examples

# A matrix plot with a mix of discrete and continuous variables
p <- ggplot(mpg) +
  geom_autopoint() +
  facet_matrix(vars(drv:fl), layer.diag = 2, grid.y.diag = FALSE)

p

# Diagonal histograms
p + geom_autohistogram()

# Diagonal density distributions
p + geom_autodensity()

# You can use them like regular layers with groupings etc
p + geom_autodensity(aes(colour = drv, fill = drv),
                     alpha = 0.4)

---

**geom_autopoint**

A point geom specialised for scatterplot matrices

Description

This geom is a specialisation of `ggplot2::geom_point()` with two changes. It defaults to mapping `x` and `y` to `.panel_x` and `.panel_y` respectively, and it defaults to using `position_auto()` to jitter the points based on the combination of position scale types.

Usage

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

Arguments

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

**stat**

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

**position**

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

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

**na.rm**

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

**show.legend**

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

**inherit.aes**

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

### See Also

- `facet_matrix` for how to lay out scatterplot matrices and `position_auto` for information about the position adjustments

### Examples

```r
# Continuous vs continuous: No jitter
ggplot(mpg) + geom_autopoint(aes(cty, hwy))

# Continuous vs discrete: sina jitter
ggplot(mpg) + geom_autopoint(aes(cty, drv))

# Discrete vs discrete: disc-jitter
ggplot(mpg) + geom_autopoint(aes(fl, drv))

# Used with facet_matrix (x and y are automatically mapped)
ggplot(mpg) +
  geom_autopoint() +
  facet_matrix(vars(drv:fl))
```

**geom_bezier**

Create quadratic or cubic bezier curves
Description

This set of geoms makes it possible to connect points creating either quadratic or cubic beziers. `bezier` and `bezier2` both work by calculating points along the bezier and connecting these to draw the curve. `bezier0` directly draws the bezier using `bezierGrob`. In line with the `geom_link()` and `geom_link2()` differences `geom_bezier` creates the points, assign an index to each interpolated point and repeat the aesthetics for the start point, while `geom_bezier2` interpolates the aesthetics between the start and end points.

Usage

```r
ggplot(data, aes(x = x, y = y)) + geom_path(position = "identity")
```

```r
stat_bezier(mapping = NULL, data = NULL, geom = "path", position = "identity", na.rm = FALSE, show.legend = NA, n = 100, inherit.aes = TRUE, ...)
```

```r
geom_bezier(mapping = NULL, data = NULL, stat = "bezier", position = "identity", arrow = NULL, lineend = "butt", na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, n = 100, ...)
```

```r
stat_bezier2(mapping = NULL, data = NULL, geom = "path_interpolate", position = "identity", na.rm = FALSE, show.legend = NA, n = 100, inherit.aes = TRUE, ...)
```

```r
geom_bezier2(mapping = NULL, data = NULL, stat = "bezier2", position = "identity", arrow = NULL, lineend = "butt", na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, n = 100, ...)
```

```r
stat_bezier0(mapping = NULL, data = NULL, geom = "bezier0", position = "identity", na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)
```

```r
geom_bezier0(mapping = NULL, data = NULL, stat = "bezier0", position = "identity", arrow = NULL, lineend = "butt", na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)
```

Arguments

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

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

**geom**
The geometric object to use display the data

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

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

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

**n**
The number of points to create for each segment

**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().

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

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

**arrow**
Arrow specification, as created by grid::arrow().

**lineend**
Line end style (round, butt, square).

**Details**
Input data is understood as a sequence of data points the first being the start point, then followed by one or two control points and then the end point. More than 4 and less than 3 points per group will throw an error. grid::bezierGrob() only takes cubic beziers so if three points are supplied the middle one as duplicated. This, along with the fact that grid::bezierGrob() estimates the curve using an x-spline means that the curves produced by geom_bezier and geom_bezier2 deviates from those produced by geom_bezier0. If you want true bezier paths use geom_bezier or geom_bezier2.

**Aesthetics**
geom_bezier, geom_bezier2 and geom_bezier0 understand the following aesthetics (required aesthetics are in bold):

- x
- y
- color
- size
- linetype
- alpha
- lineend
Computed variables

\textbf{x, y} The interpolated point coordinates

\textbf{index} The progression along the interpolation mapped between 0 and 1

Examples

```r
beziers <- data.frame(
  x = c(1, 2, 3, 4, 4, 6, 6),
  y = c(0, 2, 0, 0, 2, 2, 0),
  type = rep(c('cubic', 'quadratic'), c(3, 4)),
  point = c('end', 'control', 'end', 'end', 'control', 'control', 'end'),
  colour = letters[1:7]
)
help_lines <- data.frame(
  x = c(1, 3, 4, 6),
  xend = c(2, 2, 4, 6),
  y = 0,
  yend = 2
)

# See how control points affect the bezier
ggplot() +
  geom_segment(aes(x = x, xend = xend, y = y, yend = yend),
               data = help_lines,
               arrow = arrow(length = unit(c(0, 0, 0.5, 0.5), 'cm')),
               colour = 'grey') +
  geom_bezier(aes(x = x, y = y, group = type, linetype = type),
              data = beziers) +
  geom_point(aes(x = x, y = y, colour = point),
             data = beziers)

# geom_bezier0 is less exact
ggplot() +
  geom_segment(aes(x = x, xend = xend, y = y, yend = yend),
               data = help_lines,
               arrow = arrow(length = unit(c(0, 0, 0.5, 0.5), 'cm')),
               colour = 'grey') +
  geom_bezier0(aes(x = x, y = y, group = type, linetype = type),
               data = beziers) +
  geom_point(aes(x = x, y = y, colour = point),
             data = beziers)

# Use geom_bezier2 to interpolate between endpoint aesthetics
ggplot(beziers) +
  geom_bezier2(aes(x = x, y = y, group = type, colour = colour))
```

\textbf{geom_bsplines} \- \textit{B-splines based on control points}
Description

This set of stats and geoms makes it possible to draw b-splines based on a set of control points. As with `geom bezier()` there exists several versions each having there own strengths. The base version calculates the b-spline as a number of points along the spline and connects these with a path. The *2 version does the same but in addition interpolates aesthetics between each control point. This makes the *2 version considerably slower so it shouldn’t be used unless needed. The *0 version uses `grid::xsplineGrob()` with shape = 1 to approximate a b-spline.

Usage

```r
stat_bspline(mapping = NULL, data = NULL, geom = "path",
             position = "identity", na.rm = FALSE, n = 100, type = "clamped",
             show.legend = NA, inherit.aes = TRUE, ...)

geom_bspline(mapping = NULL, data = NULL, stat = "bspline",
             position = "identity", arrow = NULL, n = 100, type = "clamped",
             lineend = "butt", na.rm = FALSE, show.legend = NA,
             inherit.aes = TRUE, ...)

stat_bspline2(mapping = NULL, data = NULL, geom = "path_interpolate",
               position = "identity", na.rm = FALSE, n = 100, type = "clamped",
               show.legend = NA, inherit.aes = TRUE, ...)

geom_bspline2(mapping = NULL, data = NULL, stat = "bspline2",
               position = "identity", arrow = NULL, n = 100, type = "clamped",
               lineend = "butt", na.rm = FALSE, show.legend = NA,
               inherit.aes = TRUE, ...)

stat_bspline0(mapping = NULL, data = NULL, geom = "bspline0",
               position = "identity", na.rm = FALSE, show.legend = NA,
               inherit.aes = TRUE, type = "clamped", ...)

geom_bspline0(mapping = NULL, data = NULL, stat = "identity",
               position = "identity", arrow = NULL, lineend = "butt",
               na.rm = FALSE, show.legend = NA, inherit.aes = TRUE,
               type = "clamped", ...)
```

Arguments

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

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

\textbf{geom}  

The geometric object to use display the data

\textbf{position}  

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

\textbf{na.rm}  

If \texttt{FALSE}, the default, missing values are removed with a warning. If \texttt{TRUE}, missing values are silently removed.

\textbf{n}  

The number of points generated for each spline

\textbf{type}  

Either 'clamped' (default) or 'open'. The former creates a knot sequence that ensures the splines starts and ends at the terminal control points.

\textbf{show.legend}  

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

\textbf{inherit.aes}  

If \texttt{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. \texttt{borders()}.

\textbf{...}  

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

\textbf{stat}  

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

\textbf{arrow}  

Arrow specification, as created by \texttt{grid::arrow()}.

\textbf{lineend}  

Line end style (round, butt, square).

\textbf{Aesthetics}  

\texttt{geom\_bspline} understand the following aesthetics (required aesthetics are in bold):

- \texttt{x}
- \texttt{y}
- \texttt{color}
- \texttt{size}
- \texttt{linetype}
- \texttt{alpha}
- \texttt{lineend}

\textbf{Computed variables}  

\texttt{x, y} The coordinates for the path describing the spline

\texttt{index} The progression along the interpolation mapped between 0 and 1

\textbf{Author(s)}  

Thomas Lin Pedersen. The C++ code for De Boor’s algorithm has been adapted from Jason Yu-Tseh Chi implementation
geom_bspline_closed

Examples

# Define some control points
cp <- data.frame(
    x = c(
        0, -5, -5, 5, 5, 2.5, 5, 7.5, 5, 2.5, 5, 7.5, 5, -2.5, -5, -7.5, -5,
        -2.5, -5, -7.5, -5
    ),
    y = c(
        0, -5, 5, -5, 5, 5, 7.5, 5, 2.5, -5, -7.5, -5, -2.5, 5, 7.5, 5, 2.5,
        -7.5, -5, -2.5
    ),
    class = sample(letters[1:3], 21, replace = TRUE)
)

# Now create some paths between them
paths <- data.frame(
    ind = c(
        7, 5, 8, 8, 5, 9, 5, 6, 6, 5, 7, 7, 5, 1, 3, 15, 8, 5, 1, 3, 17, 9, 5,
        1, 2, 19, 6, 5, 1, 4, 12, 7, 5, 1, 4, 10, 6, 5, 1, 2, 20
    ),
    group = c(
        1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 5, 5, 5, 6, 6, 6, 7, 7,
        7, 7, 8, 8, 8, 8, 9, 9, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10
    )
)
paths$x <- cp$x[paths$ind]
paths$y <- cp$y[paths$ind]
paths$class <- cp$class[paths$ind]

ggplot(paths) +
    geom_bspline(aes(x = x, y = y, group = group, colour = ..index..)) +
    geom_point(aes(x = x, y = y), data = cp, color = 'steelblue')
ggplot(paths) +
    geom_bspline2(aes(x = x, y = y, group = group, colour = class)) +
    geom_point(aes(x = x, y = y), data = cp, color = 'steelblue')
ggplot(paths) +
    geom_bspline0(aes(x = x, y = y, group = group)) +
    geom_point(aes(x = x, y = y), data = cp, color = 'steelblue')

geom_bspline_closed  Create closed b-spline shapes

Description

This geom creates closed b-spline curves and draws them as shapes. The closed b-spline is achieved by wrapping the control points rather than the knots. The *0 version uses the grid::xsplineGrob() function with open = FALSE and can thus not be manipulated as a shape geom in the same way as the base version (expand, contract, etc).
Usage

stat_bspline_closed(mapping = NULL, data = NULL, geom = "shape",
position = "identity", na.rm = FALSE, n = 100, show.legend = NA,
inherit.aes = TRUE, ...)

geom_bspline_closed(mapping = NULL, data = NULL, stat = "bspline",
position = "identity", n = 100, na.rm = FALSE, show.legend = NA,
inherit.aes = TRUE, ...)

geom_bspline_closed0(mapping = NULL, data = NULL, stat = "identity",
position = "identity", na.rm = FALSE, show.legend = NA,
inherit.aes = TRUE, ...)

Arguments

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

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

geom The geometric object to use display the data

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

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

n The number of points generated for each spline

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().

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

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

**Aesthetics**

`geom_bspline_closed` understands the following aesthetics (required aesthetics are in bold):

- `x`
- `y`
- `color`
- `fill`
- `size`
- `linetype`
- `alpha`

**Computed variables**

- `x`, `y` The coordinates for the path describing the spline
- `index` The progression along the interpolation mapped between 0 and 1

**Author(s)**

Thomas Lin Pedersen. The C++ code for De Boor’s algorithm has been adapted from Jason Yu-Tseh Chi implementation.

**Examples**

```r
# Create 6 random control points
controls <- data.frame(
  x = runif(6),
  y = runif(6)
)

ggplot(controls, aes(x, y)) +
  geom_polygon(fill = NA, colour = "grey") +
  geom_point(colour = "red") +
  geom_bspline_closed(alpha = 0.5)
```

```
# The 0 version approximates the correct shape

ggplot(controls, aes(x, y)) +
  geom_polygon(fill = NA, colour = "grey") +
  geom_point(colour = "red") +
  geom_bspline_closed0(alpha = 0.5)
```

```
# But only the standard version supports geom_shape operations
# Be aware of self-intersections though

ggplot(controls, aes(x, y)) +
  geom_polygon(fill = NA, colour = "grey") +
  geom_point(colour = "red") +
  geom_bspline_closed(alpha = 0.5, expand = unit(2, "cm"))
```
**Description**

This set of stats and geoms makes it possible to draw circles based on a center point and a radius. In contrast to using `ggplot2::geom_point()`, the size of the circles are related to the coordinate system and not to a separate scale. These functions are intended for cartesian coordinate systems and will only produce a true circle if `ggplot2::coord_fixed()` is used.

**Usage**

```r
stat_circle(mapping = NULL, data = NULL, geom = "circle",
   position = "identity", n = 360, na.rm = FALSE, show.legend = NA,
   inherit.aes = TRUE, ...)
```

```r
geom_circle(mapping = NULL, data = NULL, stat = "circle",
   position = "identity", n = 360, expand = 0, radius = 0,
   na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)
```

**Arguments**

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

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

- **geom**
  The geometric object to use display the data

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

- **n**
  The number of points on the generated path per full circle.

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

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

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().

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

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

expand  A numeric or unit vector of length one, specifying the expansion amount. Negative values will result in contraction instead. If the value is given as a numeric it will be understood as a proportion of the plot area width.

radius  As expand but specifying the corner radius.

Aesthetics

gem_circle understand the following aesthetics (required aesthetics are in bold):

• x0
• y0
• r
• color
• fill
• size
• linetype
• alpha
• lineend

Computed variables

x, y  The start coordinates for the segment

Note

If the intend is to draw a bubble chart then use ggplot2::geom_point() and map a variable to the size scale

See Also

geom_arc_bar() for drawing arcs with fill

Examples

# Lets make some data
circles <- data.frame(
  x0 = rep(1:3, 3),
  y0 = rep(1:3, each = 3),
  r = seq(0.1, 1, length.out = 9)
)
# Behold the some circles

```r
ggplot() +
  geom_circle(aes(x0 = x0, y0 = y0, r = r, fill = r), data = circles)
```

# Use coord_fixed to ensure true circularity

```r
ggplot() +
  geom_circle(aes(x0 = x0, y0 = y0, r = r, fill = r), data = circles) +
  coord_fixed()
```

---

table:

<table>
<thead>
<tr>
<th>geom_diagonal</th>
<th>Draw horizontal diagonals</th>
</tr>
</thead>
</table>

**Description**

A diagonal is a bezier curve where the control points are moved perpendicularly towards the center in either the x or y direction a fixed amount. The versions provided here calculates horizontal diagonals meaning that the x coordinate is moved to achieve the control point. The `geom_diagonal()` and `stat_diagonal()` functions are simply helpers that takes care of calculating the position of the control points and then forwards the actual bezier calculations to `geom_bezier()`.

**Usage**

```r
stat_diagonal(mapping = NULL, data = NULL, geom = "path",
  position = "identity", n = 100, strength = 0.5, na.rm = FALSE,
  show.legend = NA, inherit.aes = TRUE, ...)
```

```r
geom_diagonal(mapping = NULL, data = NULL, stat = "diagonal",
  position = "identity", n = 100, na.rm = FALSE, strength = 0.5,
  show.legend = NA, inherit.aes = TRUE, ...)
```

```r
stat_diagonal2(mapping = NULL, data = NULL,
  geom = "path_interpolate", position = "identity", na.rm = FALSE,
  show.legend = NA, n = 100, strength = 0.5, inherit.aes = TRUE,
  ...)
```

```r
geom_diagonal2(mapping = NULL, data = NULL, stat = "diagonal2",
  position = "identity", arrow = NULL, lineend = "butt",
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, n = 100,
  strength = 0.5, ...)
```

```r
stat_diagonal0(mapping = NULL, data = NULL, geom = "bezier0",
  position = "identity", na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE, strength = 0.5, ...)
```

```r
geom_diagonal0(mapping = NULL, data = NULL, stat = "diagonal0",
  position = "identity", arrow = NULL, lineend = "butt",
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)
```
\texttt{geom_diagonal}

\begin{verbatim}
na.rm = FALSE, show.legend = NA, inherit.aes = TRUE,
strength = 0.5, ...)
\end{verbatim}

**Arguments**

- **mapping**: Set of aesthetic mappings created by \texttt{aes()} or \texttt{aes()}. If specified and \texttt{inherit.aes} = \texttt{TRUE} (the default), it is combined with the default mapping at the top level of the plot. You must supply \texttt{mapping} if there is no plot mapping.
- **data**: The data to be displayed in this layer. There are three options:
  - If \texttt{NULL}, the default, the data is inherited from the plot data as specified in the call to \texttt{ggplot}().
  - A \texttt{data.frame}, or other object, will override the plot data. All objects will be fortified to produce a data frame. See \texttt{fortify()} for which variables will be created.
  - A function will be called with a single argument, the plot data. The return value must be a \texttt{data.frame}, and will be used as the layer data. A function can be created from a formula (e.g. \texttt{~ head(.x,10)}).
- **geom**: The geometric object to use display the data
- **position**: Position adjustment, either as a string, or the result of a call to a position adjustment function.
- **n**: The number of points to create for each segment
- **strength**: The proportion to move the control point along the x-axis towards the other end of the bezier curve
- **na.rm**: If \texttt{FALSE}, the default, missing values are removed with a warning. If \texttt{TRUE}, missing values are silently removed.
- **show.legend**: logical. Should this layer be included in the legends? \texttt{NA}, the default, includes if any aesthetics are mapped. \texttt{FALSE} never includes, and \texttt{TRUE} always includes. It can also be a named logical vector to finely select the aesthetics to display.
- **inherit.aes**: If \texttt{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. \texttt{borders()}.
- **...**: Other arguments passed on to \texttt{layer()}. These are often aesthetics, used to set an aesthetic to a fixed value, like \texttt{colour = "red"} or \texttt{size = 3}. They may also be parameters to the paired geom/stat.
- **stat**: The statistical transformation to use on the data for this layer, as a string.
- **arrow**: Arrow specification, as created by \texttt{grid::arrow()}.
- **lineend**: Line end style (round, butt, square).

**Aesthetics**

\texttt{geom_diagonal} and \texttt{geom_diagonal0} understand the following aesthetics (required aesthetics are in bold):

- \texttt{x}
- \texttt{y}
geom_diagonal

- `xend`
- `yend`
- `color`
- `size`
- `linetype`
- `alpha`
- `lineend`

`geom_diagonal2` understand the following aesthetics (required aesthetics are in bold):

- `x`
- `y`
- `group`
- `color`
- `size`
- `linetype`
- `alpha`
- `lineend`

**Computed variables**

- `x, y` The interpolated point coordinates
- `index` The progression along the interpolation mapped between 0 and 1

**Examples**

```r
data <- data.frame(
  x = rep(0, 10),
  y = 1:10,
  xend = 1:10,
  yend = 2:11
)

ggplot(data) +
  geom_diagonal(aes(x, y, xend = xend, yend = yend))

# The standard version provides an index to create gradients
ggplot(data) +
  geom_diagonal(aes(x, y, xend = xend, yend = yend, alpha = stat(index)))

# The 0 version uses bezierGrob under the hood for an approximation
ggplot(data) +
  geom_diagonal0(aes(x, y, xend = xend, yend = yend))

# The 2 version allows you to interpolate between endpoint aesthetics
data2 <- data.frame(
  x = c(data$x, data$xend),
  y = c(data$y, data$yend),
  xend = c(data$xend, data2$xend),
  yend = c(data$yend, data2$yend)
)

ggplot(data2) +
  geom_line(aes(x, y, xend = xend, yend = yend, alpha = index))
```

```r
```

```r
```
# geom_diagonal_wide

Draw an area defined by an upper and lower diagonal

**Description**

The `geom_diagonal_wide()` function draws a thick diagonal, that is, a polygon confined between a lower and upper diagonal. As with the diagonal functions in ggforce, the wide diagonal variant is horizontal.

**Usage**

```r
stat_diagonal_wide(mapping = NULL, data = NULL, geom = "shape", position = "identity", n = 100, strength = 0.5, na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)
```

```r
gem_diagonal_wide(mapping = NULL, data = NULL, stat = "diagonal_wide", position = "identity", n = 100, na.rm = FALSE, strength = 0.5, show.legend = NA, inherit.aes = TRUE, ...)
```

**Arguments**

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

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

geom
The geometric object to use display the data

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

n
The number of points to create for each of the bounding diagonals

strength
The proportion to move the control point along the x-axis towards the other end of the bezier curve

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

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

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

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

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

Aesthetics

geom_diagonal_wide understand the following aesthetics (required aesthetics are in bold):

• x
• y
• group
• color
• size
• linetype
• alpha
• lineend

Examples

data <- data.frame(
  x = c(1, 2, 2, 1, 2, 3, 3, 2),
  y = c(1, 2, 3, 2, 3, 1, 2, 5),
  group = c(1, 1, 1, 1, 2, 2, 2, 2)
)

ggplot(data) +
  geom_diagonal_wide(aes(x, y, group = group))

# The strength control the steepness

 ggplot(data, aes(x, y, group = group)) +
  geom_diagonal_wide(strength = 0.75, alpha = 0.5, fill = 'red') +
geom_diagonal_wide(strength = 0.25, alpha = 0.5, fill = 'blue')

# The diagonal_wide geom uses geom_shape under the hood, so corner rounding
# etc are all there

ggplot(data) +
  geom_diagonal_wide(aes(x, y, group = group), radius = unit(5, 'mm'))

---

**geom_ellipse**

*Draw (super)ellipses based on the coordinate system scale*

**Description**

This is a generalisation of `geom_circle()` that allows you to draw ellipses at a specified angle and center relative to the coordinate system. Apart from letting you draw regular ellipsis, the stat is using the generalised formula for superellipses which can be utilised by setting the m1 and m2 aesthetics. If you only set the m1 the m2 value will follow that to ensure a symmetric appearance.

**Usage**

```r
stat_ellip(mapping = NULL, data = NULL, geom = "circle",
position = "identity", n = 360, na.rm = FALSE, show.legend = NA,
inherit.aes = TRUE, ...)
```

```r
geom_ellipse(mapping = NULL, data = NULL, stat = "ellip",
position = "identity", n = 360, na.rm = FALSE, show.legend = NA,
inherit.aes = TRUE, ...)
```

**Arguments**

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

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

- **geom**
  - The geometric object to use display the data

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

- **n**
  - The number of points to sample along the ellipse.
na.rm If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

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

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

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

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

**Aesthetics**

`geom_ellipse` understands the following aesthetics (required aesthetics are in bold):

- `x0`
- `y0`
- `a`
- `b`
- `angle`
- `m1`
- `m2`
- `color`
- `fill`
- `size`
- `linetype`
- `alpha`
- `lineend`

**Computed variables**

- `x, y` The coordinates for the points along the ellipse

**Examples**

```r
# Basic usage
ggplot() +
  geom_ellipse(aes(x0 = 0, y0 = 0, a = 10, b = 3, angle = 0)) +
  coord_fixed()
```

```r
# Rotation
# Note that it expects radians and rotates the ellipse counter-clockwise
ggplot() +
```
```r
geom_ellipse(aes(x0 = 0, y0 = 0, a = 10, b = 3, angle = pi / 4)) +
  coord_fixed()

# Draw a super ellipse
ggplot() +
  geom_ellipse(aes(x0 = 0, y0 = 0, a = 6, b = 3, angle = -pi / 3, m1 = 3)) +
  coord_fixed()
```

---

**geom_link**  
*Link points with paths*

**Description**

This set of geoms makes it possible to connect points using straight lines. Before you think `ggplot2::geom_segment()` and `ggplot2::geom_path()`, these functions have some additional tricks up their sleeves. `geom_link` connects two points in the same way as `ggplot2::geom_segment()` but does so by interpolating multiple points between the two. An additional column called `index` is added to the data with a sequential progression of the interpolated points. This can be used to map color or size to the direction of the link. `geom_link2` uses the same syntax as `ggplot2::geom_path()` but interpolates between the aesthetics given by each row in the data.

**Usage**

```r
stat_link(mapping = NULL, data = NULL, geom = "path",
  position = "identity", na.rm = FALSE, show.legend = NA, n = 100,
  inherit.aes = TRUE, ...)

stat_link2(mapping = NULL, data = NULL, geom = "path_interpolate",
  position = "identity", na.rm = FALSE, show.legend = NA, n = 100,
  inherit.aes = TRUE, ...)

geom_link(mapping = NULL, data = NULL, stat = "link",
  position = "identity", arrow = NULL, lineend = "butt",
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, n = 100,
  ...)  

geom_link2(mapping = NULL, data = NULL, stat = "link2",
  position = "identity", arrow = NULL, lineend = "butt",
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, n = 100,
  ...)  

geom_link0(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", ..., arrow = NULL, arrow.fill = NULL,
  lineend = "butt", linejoin = "round", na.rm = FALSE,
  show.legend = NA, inherit.aes = TRUE)
```
Arguments

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

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

geom  The geometric object to use display the data

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

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

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

n  The number of points to create for each segment

inherit.aes  If \texttt{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. \texttt{borders()}.

...  Other arguments passed on to \texttt{layer()}. These are often aesthetics, used to set an aesthetic to a fixed value, like \texttt{colour = “red”} or \texttt{size = 3}. They may also be parameters to the paired geom/stat.

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

arrow  Arrow specification, as created by \texttt{grid::arrow()}.

lineend  Line end style (round, butt, square).

arrow.fill  fill colour to use for the arrow head (if closed). \texttt{NULL} means use \texttt{colour} aesthetic.

linejoin  Line join style (round, mitre, bevel).

Aesthetics

\texttt{geom\_link} understand the following aesthetics (required aesthetics are in bold):

• \texttt{x}
• \texttt{y}
• \texttt{xend}
• \texttt{yend}
• color
• size
• linetype
• alpha
• lineend

geom_link2 understand the following aesthetics (required aesthetics are in bold):

• x
• y
• color
• size
• linetype
• alpha
• lineend

Computed variables

x, y  The interpolated point coordinates
index The progression along the interpolation mapped between 0 and 1

Examples

# Lets make some data
lines <- data.frame(
  x = c(5, 12, 15, 9, 6),
  y = c(17, 20, 4, 15, 5),
  xend = c(19, 17, 2, 9, 5),
  yend = c(10, 18, 7, 12, 1),
  width = c(1, 10, 6, 2, 3),
  colour = letters[1:5]
)

ggplot(lines) +
  geom_link(aes(x = x, y = y, xend = xend, yend = yend, colour = colour,
                alpha = stat(index), size = stat(index)))

ggplot(lines) +
  geom_link2(aes(x = x, y = y, colour = colour, size = width, group = 1),
             lineend = 'round', n = 500)

# geom_link0 is simply an alias for geom_segment to put the link geoms in
# line with the other line geoms with multiple versions. `index` is not
# available here
ggplot(lines) +
  geom_link0(aes(x = x, y = y, xend = xend, yend = yend, colour = colour))
geom_mark_circle

Annotate areas with circles

description

This geom lets you annotate sets of points via circles. The enclosing circles are calculated at draw
time and the most optimal enclosure at the given aspect ratio is thus guaranteed. As with the
other geom_mark_* geoms the enclosure inherits from geom_shape() and defaults to be expanded
slightly to better enclose the points.

usage

geom_mark_circle(mapping = NULL, data = NULL, stat = "identity",
position = "identity", expand = unit(5, "mm"), radius = expand,
n = 100, label.margin = margin(2, 2, 2, 2, "mm"),
label.width = NULL, label.minwidth = unit(50, "mm"),
label.hjust = 0, label.fontsize = 12, label.family = "",
label.lineheight = 1, label.fontface = c("bold", "plain"),
lable.fill = "white", label.colour = "black",
label.buffer = unit(10, "mm"), con.colour = "black",
con.size = 0.5, con.type = "elbow", con.linetype = 1,
con.border = "one", con.cap = unit(3, "mm"), con.arrow = NULL, ...
na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)

arguments

mapping Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data The data to be displayed in this layer. There are three options:
If NULL, the default, the data is inherited from the plot data as specified in the
call to ggplot().
A data.frame, or other object, will override the plot data. All objects will be
fortified to produce a data frame. See fortify() for which variables will be
created.
A function will be called with a single argument, the plot data. The return
value must be a data.frame, and will be used as the layer data. A function
can be created from a formula (e.g. ~ head(.x,10)).
stat The statistical transformation to use on the data for this layer, as a string.
position Position adjustment, either as a string, or the result of a call to a position adjust-
ment function.
expand A numeric or unit vector of length one, specifying the expansion amount. Neg-
ative values will result in contraction instead. If the value is given as a numeric
it will be understood as a proportion of the plot area width.
radius As expand but specifying the corner radius.
**geom_mark_circle**

- **n**: The number of points used to draw each circle. Defaults to 100.
- **label.margin**: The margin around the annotation boxes, given by a call to `ggplot2::margin()`.
- **label.width**: A fixed width for the label. Set to `NULL` to let the text or `label.minwidth` decide.
- **label.minwidth**: The minimum width to provide for the description. If the size of the label exceeds this, the description is allowed to fill as much as the label.
- **label.hjust**: The horizontal justification for the annotation. If it contains two elements the first will be used for the label and the second for the description.
- **label.fontsize**: The size of the text for the annotation. If it contains two elements the first will be used for the label and the second for the description.
- **label.family**: The font family used for the annotation. If it contains two elements the first will be used for the label and the second for the description.
- **label.lineheight**: The height of a line as a multiplier of the fontsize. If it contains two elements the first will be used for the label and the second for the description.
- **label.fontface**: The font face used for the annotation. If it contains two elements the first will be used for the label and the second for the description.
- **label.fill**: The fill colour for the annotation box.
- **label.colour**: The text colour for the annotation. If it contains two elements the first will be used for the label and the second for the description.
- **label.buffer**: The size of the region around the mark where labels cannot be placed.
- **con.colour**: The colour for the line connecting the annotation to the mark.
- **con.size**: The width of the connector.
- **con.type**: The type of the connector. Either "elbow", "straight", or "none".
- **con.linetype**: The linetype of the connector.
- **con.border**: The bordertype of the connector. Either "one" (to draw a line on the horizontal side closest to the mark), "all" (to draw a border on all sides), or "none" (not going to explain that one).
- **con.cap**: The distance before the mark that the line should stop at.
- **con.arrow**: An arrow specification for the connection using `grid::arrow()` for the end pointing towards the mark.
- **...**: Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.
- **na.rm**: If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.
- **show.legend**: logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.
- **inherit.aes**: If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders()`.
Annotation

All `geom_mark_*` allows you to put descriptive textboxes connected to the mark on the plot, using the `label` and `description` aesthetics. The textboxes are automatically placed close to the mark, but without obscuring any of the datapoints in the layer. The placement is dynamic so if you resize the plot you'll see that the annotation might move around as areas become big enough or too small to fit the annotation. If there's not enough space for the annotation without overlapping data it will not get drawn. In these cases try resizing the plot, change the size of the annotation, or decrease the buffer region around the marks.

Filtering

Often marks are used to draw attention to, or annotate specific features of the plot and it is thus not desirable to have marks around everything. While it is possible to simply pre-filter the data used for the mark layer, the `geom_mark_*` geoms also comes with a dedicated `filter` aesthetic that, if set, will remove all rows where it evaluates to `FALSE`. There are multiple benefits of using this instead of prefiltering. First, you don’t have to change your data source, making your code more adaptable for exploration. Second, the data removed by the filter aesthetic is remembered by the geom, and any annotation will take care not to overlap with the removed data.

Aesthetics

`geom_mark_circle` understand the following aesthetics (required aesthetics are in bold):

- `x`
- `y`
- `filter`
- `label`
- `description`
- `color`
- `fill`
- `group`
- `size`
- `linetype`
- `alpha`

See Also

Other mark geoms: `geom_mark_ellipse`, `geom_mark_hull`, `geom_mark_rect`

Examples

```r
ggplot(iris, aes(Petal.Length, Petal.Width)) +
  geom_mark_circle(aes(fill = Species, filter = Species != 'versicolor')) +
  geom_point()
```

# Add annotation
```r
ggplot(iris, aes(Petal.Length, Petal.Width)) +
```
geom_mark_ellipse

Annotate areas with ellipses

Description

This geom lets you annotate sets of points via ellipses. The enclosing ellipses are estimated using the Khachiyan algorithm which guarantees and optimal solution within the given tolerance level. As this geom is often expanded it is of lesser concern that some points are slightly outside the ellipsis. The Khachiyan algorithm has polynomial complexity and can thus suffer from scaling issues. Still, it is only calculated on the convex hull of the groups, so performance issues should be rare (it can easily handle a hull consisting of 1000 points).

Usage

geom_mark_ellipse(mapping = NULL, data = NULL, stat = "identity", position = "identity", expand = unit(5, "mm"), radius = expand, n = 100, tol = 0.01, label.margin = margin(2, 2, 2, 2, "mm"), label.width = NULL, label.minwidth = unit(50, "mm"), label.hjust = 0, label.fontsize = 12, label.family = "", label.lineheight = 1, label.fontface = c("bold", "plain"),

# Long descriptions are automatically wrapped to fit into the width
iris$desc <- c('A super Iris - and it knows it',
'Pretty mediocre Iris, but give it a couple of years and it might surprise you',
"You'll never guess what this Iris does every Sunday"
)[iris$Species]

ggplot(iris, aes(Petal.Length, Petal.Width)) +
geom_mark_circle(aes(fill = Species, label = Species, description = desc,
    filter = Species == 'setosa')) +
ggplot(iris, aes(Petal.Length, Petal.Width)) +
geom_mark_circle(aes(fill = Species, label = Species, description = desc,
    filter = Species == 'setosa')) +
ggplot(iris, aes(Petal.Length, Petal.Width)) +
geom_mark_circle(aes(fill = Species, label = Species),
    label.buffer = unit(30, 'mm')) +
ggplot(iris, aes(Petal.Length, Petal.Width)) +
geom_mark_circle(aes(fill = Species, label = Species),
    label.buffer = unit(30, 'mm')) +
ggplot(iris, aes(Petal.Length, Petal.Width)) +
geom_mark_circle(aes(fill = Species, label = Species),
    con.cap = 0) +
ggplot(iris, aes(Petal.Length, Petal.Width)) +
geom_mark_circle(aes(fill = Species, label = Species),
    con.cap = 0)

# Change the buffer size to move labels farther away (or closer) from the
# marks

ggplot(iris, aes(Petal.Length, Petal.Width)) +
geom_mark_circle(aes(fill = Species, label = Species),
    label.buffer = unit(30, 'mm')) +
ggplot(iris, aes(Petal.Length, Petal.Width)) +
geom_mark_circle(aes(fill = Species, label = Species),
    label.buffer = unit(30, 'mm')) +
ggplot(iris, aes(Petal.Length, Petal.Width)) +
geom_mark_circle(aes(fill = Species, label = Species),
    con.cap = 0) +

desc = c('
A super Iris - and it knows it',
'Pretty mediocre Iris, but give it a couple of years and it might surprise you',
"You'll never guess what this Iris does every Sunday"
)[iris$Species]
label.fill = "white", label.colour = "black",
label.buffer = unit(10, "mm"), con.colour = "black",
con.size = 0.5, con.type = "elbow", con.linetype = 1,
con.border = "one", con.cap = unit(3, "mm"), con.arrow = NULL, ...
na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)

Arguments

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

stat The statistical transformation to use on the data for this layer, as a string.
position Position adjustment, either as a string, or the result of a call to a position adjust-
ment function.
expand A numeric or unit vector of length one, specifying the expansion amount. Negative values will result in contraction instead. If the value is given as a numeric
it will be understood as a proportion of the plot area width.
radius As expand but specifying the corner radius.
n The number of points used to draw each circle. Defaults to 100
tol The tolerance cutoff. Lower values will result in ellipses closer to the optimal solution. Defaults to 0.01

label.margin The margin around the annotation boxes, given by a call to `ggplot2::margin()`
label.width A fixed width for the label. Set to NULL to let the text or label.minwidth decide
label.minwidth The minimum width to provide for the description. If the size of the label ex-
ceeds this, the the description is allowed to fill as much as the label
label.hjust The horizontal justification for the annotation. If it contains two elements the first will be used for the label and the second for the description.
label.fontsize The size of the text for the annotation. If it contains two elements the first will be used for the label and the second for the description.
label.family The font family used for the annotation. If it contains two elements the first will be used for the label and the second for the description.
label.lineheight The height of a line as a multiplier of the fontsize. If it contains two elements the first will be used for the label and the second for the description.
The font face used for the annotation. If it contains two elements the first will be used for the label and the second for the description.

The fill colour for the annotation box.

The text colour for the annotation. If it contains two elements the first will be used for the label and the second for the description.

The size of the region around the mark where labels cannot be placed.

The colour for the line connecting the annotation to the mark

The width of the connector

The type of the connector. Either "elbow", "straight", or "none".

The linetype of the connector

The bordertype of the connector. Either "one" (to draw a line on the horizontal side closest to the mark), "all" (to draw a border on all sides), or "none" (not going to explain that one)

The distance before the mark that the line should stop at.

An arrow specification for the connection using `grid::arrow()` for the end pointing towards the mark

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

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

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.

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()`.

**Aesthetics**

`geom_mark_ellipse` understand the following aesthetics (required aesthetics are in bold):

- `x`
- `y`
- `filter`
- `label`
- `description`
- `color`
- `fill`
- `group`
- `size`
- `linetype`
- `alpha`
Annotation

All geom_mark_* allows you to put descriptive textboxes connected to the mark on the plot, using the label and description aesthetics. The textboxes are automatically placed close to the mark, but without obscuring any of the datapoints in the layer. The placement is dynamic so if you resize the plot you’ll see that the annotation might move around as areas become big enough or too small to fit the annotation. If there’s not enough space for the annotation without overlapping data it will not get drawn. In these cases try resizing the plot, change the size of the annotation, or decrease the buffer region around the marks.

Filtering

Often marks are used to draw attention to, or annotate specific features of the plot and it is thus not desirable to have marks around everything. While it is possible to simply pre-filter the data used for the mark layer, the geom_mark_* geoms also comes with a dedicated filter aesthetic that, if set, will remove all rows where it evaluatates to FALSE. There are multiple benefits of using this instead of prefiltering. First, you don’t have to change your data source, making your code more adaptable for exploration. Second, the data removed by the filter aesthetic is remembered by the geom, and any annotation will take care not to overlap with the removed data.

See Also

Other mark geoms: geom_mark_circle, geom_mark_hull, geom_mark_rect

Examples

ggplot(iris, aes(Petal.Length, Petal.Width)) +
  geom_mark_ellipse(aes(fill = Species, filter = Species != 'versicolor')) +
  geom_point()

# Add annotation
ggplot(iris, aes(Petal.Length, Petal.Width)) +
  geom_mark_ellipse(aes(fill = Species, label = Species)) +
  geom_point()

# Long descriptions are automatically wrapped to fit into the width
iris$desc <- c(
  'A super Iris - and it knows it',
  'Pretty mediocre Iris, but give it a couple of years and it might surprise you',
  'You’ll never guess what this Iris does every Sunday'
)[iris$Species]

  geom_mark_ellipse(aes(fill = Species, label = Species, description = desc,
              filter = Species == 'setosa')) +
  geom_point()

# Change the buffer size to move labels farther away (or closer) from the # marks
  geom_mark_ellipse(aes(fill = Species, label = Species),
              label.buffer = unit(40, 'mm')) +
geom_point()

# The connector is capped a bit before it reaches the mark, but this can be
# controlled
ggplot(iris, aes(Petal.Length, Petal.Width)) +
  geom_mark_ellipse(aes(fill = Species, label = Species),
    con.cap = 0) +
  geom_point()

---

**geom_mark_hull**

Annotate areas with hulls

**Description**

This geom lets you annotate sets of points via hulls. While convex hulls are most common due to
their clear definition, they can lead to large areas covered that does not contain points. Due to this
geom_mark_hull uses concaveman which lets you adjust concavity of the resulting hull. The hull
is calculated at draw time, and can thus change as you resize the plot. In order to clearly contain all
points, and for aesthetic purpose the resulting hull is expanded 5mm and rounded on the corners.
This can be adjusted with the expand and radius parameters.

**Usage**

```r
geom_mark_hull(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", expand = unit(5, "mm"), radius = unit(2.5, "mm"),
  concavity = 2, label.margin = margin(2, 2, 2, 2, "mm"),
  label.width = NULL, label.minwidth = unit(50, "mm"),
  label.hjust = 0, label.fontsize = 12, label.family = "",
  label.lineheight = 1, label.fontface = c("bold", "plain"),
  label.fill = "white", label.colour = "black",
  label.buffer = unit(10, "mm"), con.colour = "black",
  con.size = 0.5, con.type = "elbow", con.linetype = 1,
  con.border = "one", con.cap = unit(3, "mm"), con.arrow = NULL, ...
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```

**Arguments**

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

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

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

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

**expand**  
A numeric or unit vector of length one, specifying the expansion amount. Negative values will result in contraction instead. If the value is given as a numeric it will be understood as a proportion of the plot area width.

**radius**  
As expand but specifying the corner radius.

**concavity**  
A measure of the concavity of the hull. 1 is very concave while it approaches convex as it grows. Defaults to 2

**label.margin**  
The margin around the annotation boxes, given by a call to `ggplot2::margin()`

**label.width**  
A fixed width for the label. Set to NULL to let the text or label.minwidth decide

**label.minwidth**  
The minimum width to provide for the description. If the size of the label exceeds this, the the description is allowed to fill as much as the label

**label.hjust**  
The horizontal justification for the annotation. If it contains two elements the first will be used for the label and the second for the description.

**label.fontsize**  
The size of the text for the annotation. If it contains two elements the first will be used for the label and the second for the description.

**label.family**  
The font family used for the annotation. If it contains two elements the first will be used for the label and the second for the description.

**label.lineheight**  
The height of a line as a multiplier of the fontsize. If it contains two elements the first will be used for the label and the second for the description.

**label.fontface**  
The font face used for the annotation. If it contains two elements the first will be used for the label and the second for the description.

**label.fill**  
The fill colour for the annotation box.

**label.colour**  
The text colour for the annotation. If it contains two elements the first will be used for the label and the second for the description.

**label.buffer**  
The size of the region around the mark where labels cannot be placed.

**con.colour**  
The colour for the line connecting the annotation to the mark

**con.size**  
The width of the connector

**con.type**  
The type of the connector. Either "elbow", "straight", or "none".

**con.linetype**  
The linetype of the connector

**con.border**  
The border type of the connector. Either "one" (to draw a line on the horizontal side closest to the mark), "all" (to draw a border on all sides), or "none" (not going to explain that one)

**con.cap**  
The distance before the mark that the line should stop at.

**con.arrow**  
An arrow specification for the connection using `grid::arrow()` for the end pointing towards the mark.
geom_mark_hull

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

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

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

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

Aesthetics

geom_mark_hull understand the following aesthetics (required aesthetics are in bold):

• x
• y
• filter
• label
• description
• color
• fill
• group
• size
• linetype
• alpha

Annotation

All geom_mark_* allows you to put descriptive textboxes connected to the mark on the plot, using the label and description aesthetics. The textboxes are automatically placed close to the mark, but without obscuring any of the datapoints in the layer. The placement is dynamic so if you resize the plot you’ll see that the annotation might move around as areas become big enough or too small to fit the annotation. If there’s not enough space for the annotation without overlapping data it will not get drawn. In these cases try resizing the plot, change the size of the annotation, or decrease the buffer region around the marks.

Filtering

Often marks are used to draw attention to, or annotate specific features of the plot and it is thus not desirable to have marks around everything. While it is possible to simply pre-filter the data used for the mark layer, the geom_mark_* geoms also comes with a dedicated filter aesthetic that, if set, will remove all rows where it evaluates to FALSE. There are multiple benefits of using this instead of prefiltering. First, you don’t have to change your data source, making your code more adaptable for exploration. Second, the data removed by the filter aesthetic is remembered by the geom, and any annotation will take care not to overlap with the removed data.
See Also

Other mark geoms: `geom_mark_circle, geom_mark_ellipse, geom_mark_rect`

Examples

```r
## requires the concaveman packages
ggplot(iris, aes(Petal.Length, Petal.Width)) +
  geom_mark_hull(aes(fill = Species, filter = Species != 'versicolor')) +
  geom_point()

# Adjusting the concavity lets you change the shape of the hull
ggplot(iris, aes(Petal.Length, Petal.Width)) +
  geom_mark_hull(aes(fill = Species, filter = Species != 'versicolor'),
                 concavity = 1)
  geom_point()

ggplot(iris, aes(Petal.Length, Petal.Width)) +
  geom_mark_hull(aes(fill = Species, filter = Species != 'versicolor'),
                 concavity = 10)
  geom_point()

# Add annotation
ggplot(iris, aes(Petal.Length, Petal.Width)) +
  geom_mark_hull(aes(fill = Species, label = Species)) +
  geom_point()

# Long descriptions are automatically wrapped to fit into the width
iris$desc <- c('A super Iris - and it knows it',
               'Pretty mediocre Iris, but give it a couple of years and it might surprise you',
               'You'll never guess what this Iris does every Sunday')

iris$Species

ggplot(iris, aes(Petal.Length, Petal.Width)) +
  geom_mark_hull(aes(fill = Species, label = Species, description = desc,
                      filter = Species == 'setosa')) +
  geom_point()

# Change the buffer size to move labels farther away (or closer) from the
# marks
ggplot(iris, aes(Petal.Length, Petal.Width)) +
  geom_mark_hull(aes(fill = Species, label = Species),
                 label.buffer = unit(40, 'mm'))
  geom_point()

# The connector is capped a bit before it reaches the mark, but this can be
# controlled
ggplot(iris, aes(Petal.Length, Petal.Width)) +
  geom_mark_hull(aes(fill = Species, label = Species),
                 con.cap = 0)
```
Description

This geom lets you annotate sets of points via rectangles. The rectangles are simply scaled to the range of the data and as with the other geom_mark_*() geoms expanded and have rounded corners.

Usage

```r
geom_mark_rect(mapping = NULL, data = NULL, stat = "identity", position = "identity", expand = unit(5, "mm"), radius = unit(2.5, "mm"), label.margin = margin(2, 2, 2, 2, "mm"), label.width = NULL, label.minwidth = unit(50, "mm"), label.hjust = 0, label.fontsize = 12, label.family = "", label.lineheight = 1, label.fontface = c("bold", "plain"), label.fill = "white", label.colour = "black", label.buffer = unit(10, "mm"), con.colour = "black", con.size = 0.5, con.type = "elbow", con.linetype = 1, con.border = "one", con.cap = unit(3, "mm"), con.arrow = NULL, ..., na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```

Arguments

- **mapping**: Set of aesthetic mappings created by `aes()` or `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
- **data**: The data to be displayed in this layer. There are three options:
  - If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
  - A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
  - A function will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)).
- **stat**: The statistical transformation to use on the data for this layer, as a string.
- **position**: Position adjustment, either as a string, or the result of a call to a position adjustment function.
- **expand**: A numeric or unit vector of length one, specifying the expansion amount. Negative values will result in contraction instead. If the value is given as a numeric it will be understood as a proportion of the plot area width.
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>radius</td>
<td>As expand but specifying the corner radius.</td>
</tr>
<tr>
<td>label.margin</td>
<td>The margin around the annotation boxes, given by a call to <code>ggplot2::margin()</code></td>
</tr>
<tr>
<td>label.width</td>
<td>A fixed width for the label. Set to NULL to let the text or label.minwidth decide</td>
</tr>
<tr>
<td>label.minwidth</td>
<td>The minimum width to provide for the description. If the size of the label exceeds this, the description is allowed to fill as much as the label</td>
</tr>
<tr>
<td>label.hjust</td>
<td>The horizontal justification for the annotation. If it contains two elements the first will be used for the label and the second for the description.</td>
</tr>
<tr>
<td>label.fontsize</td>
<td>The size of the text for the annotation. If it contains two elements the first will be used for the label and the second for the description.</td>
</tr>
<tr>
<td>label.family</td>
<td>The font family used for the annotation. If it contains two elements the first will be used for the label and the second for the description.</td>
</tr>
<tr>
<td>label.lineheight</td>
<td>The height of a line as a multiplier of the fontsize. If it contains two elements the first will be used for the label and the second for the description.</td>
</tr>
<tr>
<td>label.fontface</td>
<td>The font face used for the annotation. If it contains two elements the first will be used for the label and the second for the description.</td>
</tr>
<tr>
<td>label.fill</td>
<td>The fill colour for the annotation box.</td>
</tr>
<tr>
<td>label.colour</td>
<td>The text colour for the annotation. If it contains two elements the first will be used for the label and the second for the description.</td>
</tr>
<tr>
<td>label.buffer</td>
<td>The size of the region around the mark where labels cannot be placed.</td>
</tr>
<tr>
<td>con.colour</td>
<td>The colour for the line connecting the annotation to the mark</td>
</tr>
<tr>
<td>con.size</td>
<td>The width of the connector</td>
</tr>
<tr>
<td>con.type</td>
<td>The type of the connector. Either &quot;elbow&quot;, &quot;straight&quot;, or &quot;none&quot;.</td>
</tr>
<tr>
<td>con.linetype</td>
<td>The linetype of the connector</td>
</tr>
<tr>
<td>con.border</td>
<td>The bordertype of the connector. Either &quot;one&quot; (to draw a line on the horizontal side closest to the mark), &quot;all&quot; (to draw a border on all sides), or &quot;none&quot; (not going to explain that one)</td>
</tr>
<tr>
<td>con.cap</td>
<td>The distance before the mark that the line should stop at.</td>
</tr>
<tr>
<td>con.arrow</td>
<td>An arrow specification for the connection using <code>grid::arrow()</code> for the end pointing towards the mark</td>
</tr>
<tr>
<td>...</td>
<td>Other arguments passed on to <code>layer()</code>. These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = &quot;red&quot;</code> or <code>size = 3</code>. They may also be parameters to the paired geom/stat.</td>
</tr>
<tr>
<td>na.rm</td>
<td>If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.</td>
</tr>
<tr>
<td>show.legend</td>
<td>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.</td>
</tr>
<tr>
<td>inherit.aes</td>
<td>If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. <code>borders()</code></td>
</tr>
</tbody>
</table>
geom_mark_rect

Aesthetics

geom_mark_rect understand the following aesthetics (required aesthetics are in bold):

• x
• y
• filter
• label
• description
• color
• fill
• group
• size
• linetype
• alpha

Annotation

All geom_mark_* allows you to put descriptive textboxes connected to the mark on the plot, using the label and description aesthetics. The textboxes are automatically placed close to the mark, but without obscuring any of the datapoints in the layer. The placement is dynamic so if you resize the plot you’ll see that the annotation might move around as areas become big enough or too small to fit the annotation. If there’s not enough space for the annotation without overlapping data it will not get drawn. In these cases try resizing the plot, change the size of the annotation, or decrease the buffer region around the marks.

Filtering

Often marks are used to draw attention to, or annotate specific features of the plot and it is thus not desirable to have marks around everything. While it is possible to simply pre-filter the data used for the mark layer, the geom_mark_* geoms also comes with a dedicated filter aesthetic that, if set, will remove all rows where it evaluates to FALSE. There are multiple benefits of using this instead of prefiltering. First, you don’t have to change your data source, making your code more adaptable for exploration. Second, the data removed by the filter aesthetic is remembered by the geom, and any annotation will take care not to overlap with the removed data.

See Also

Other mark geoms: geom_mark_circle, geom_mark_ellipse, geom_mark_hull

Examples

ggplot(iris, aes(Petal.Length, Petal.Width)) +
gem_mark_rect(aes(fill = Species, filter = Species != 'versicolor')) +
gem_point()

# Add annotation
ggplot(iris, aes(Petal.Length, Petal.Width)) +
geom_mark_rect(aes(fill = Species, label = Species)) +
geom_point()

# Long descriptions are automatically wrapped to fit into the width
iris$desc <- c(
'A super Iris - and it knows it',
'Pretty mediocre Iris, but give it a couple of years and it might surprise you',
'You'll never guess what this Iris does every Sunday'
)[iris$Species]

ggplot(iris, aes(Petal.Length, Petal.Width)) +
geom_mark_rect(aes(fill = Species, label = Species, description = desc,
  filter = Species == 'setosa')) +
geom_point()

# Change the buffer size to move labels farther away (or closer) from the
# marks
ggplot(iris, aes(Petal.Length, Petal.Width)) +
geom_mark_rect(aes(fill = Species, label = Species),
  label.buffer = unit(30, 'mm')) +
geom_point()

# The connector is capped a bit before it reaches the mark, but this can be
# controlled
ggplot(iris, aes(Petal.Length, Petal.Width)) +
geom_mark_rect(aes(fill = Species, label = Species),
  con.cap = 0) +
geom_point()

---

**geom_parallel_sets**

*Create Parallel Sets diagrams*

**Description**

A parallel sets diagram is a type of visualisation showing the interaction between multiple categorical variables. If the variables has an intrinsic order the representation can be thought of as a Sankey Diagram. If each variable is a point in time it will resemble an alluvial diagram.

**Usage**

```r
stat_parallel_sets(mapping = NULL, data = NULL, geom = "shape",
position = "identity", n = 100, strength = 0.5, sep = 0.05,
axis.width = 0, na.rm = FALSE, show.legend = NA,
inherit.aes = TRUE, ...)
```

```r
gem_parallel_sets(mapping = NULL, data = NULL,
stat = "parallel_sets", position = "identity", n = 100,
na.rm = FALSE, sep = 0.05, strength = 0.5, axis.width = 0,
show.legend = NA, inherit.aes = TRUE, ...)
```
stat_parallel_sets_axes(mapping = NULL, data = NULL, 
geom = "parallel_sets_axes", position = "identity", sep = 0.05, 
axis.width = 0, na.rm = FALSE, show.legend = NA, 
inherit.aes = TRUE, ...)

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

geom_parallel_sets_labels(mapping = NULL, data = NULL, 
stat = "parallel_sets_axes", angle = -90, position = "identity", 
na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)

Arguments

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

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

geom
The geometric object to use display the data

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

n
The number of points to create for each of the bounding diagonals

strength
The proportion to move the control point along the x-axis towards the other end of the bezier curve

sep
The proportional separation between categories within a variable

axis.width
The width of the area around each variable axis

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

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

inherit.aes
If FALSE, overrides the default aesthetics, rather than combining with them.
This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. borders().
Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

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

**angle**
The angle of the axis label text

---

**Details**

In a parallel sets visualization each categorical variable will be assigned a position on the x-axis. The size of the intersection of categories from neighboring variables are then shown as thick diagonals, scaled by the sum of elements shared between the two categories. The natural data representation for such as plot is to have each categorical variable in a separate column and then have a column giving the amount/magnitude of the combination of levels in the row. This representation is unfortunately not fitting for the `ggplot2` API which needs every position encoding in the same column. To make it easier to work with `ggforce` provides a helper `gather_set_data()`, which takes care of the transformation.

**Aesthetics**

`geom_parallel_sets` understand the following aesthetics (required aesthetics are in bold):

- `x`
- `id`
- `split`
- `value`
- `color`
- `fill`
- `size`
- `linetype`
- `alpha`
- `lineend`

**Author(s)**

Thomas Lin Pedersen

**Examples**

```r
data <- reshape2::melt(Titanic)
data <- gather_set_data(data, 1:4)

ggplot(data, aes(x, id = id, split = y, value = value)) +
  geom_parallel_sets(aes(fill = Sex), alpha = 0.3, axis.width = 0.1) +
  geom_parallel_sets_axes(axis.width = 0.1) +
  geom_parallel_sets_labels(colour = "white")
```
**geom_regon**  
*Draw regular polygons by specifying number of sides*

**Description**  
This geom makes it easy to construct regular polygons (polygons where all sides and angles are equal) by specifying the number of sides, position, and size. The polygons are always rotated so that they "rest" on a flat side, but this can be changed with the angle aesthetic. The size is based on the radius of their circumcircle and is thus not proportional to their area.

**Usage**

```r
stat_regon(mapping = NULL, data = NULL, geom = "shape",  
position = "identity", na.rm = FALSE, show.legend = NA,  
inherit.aes = TRUE, ...)
```

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

**Arguments**

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

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

- **geom**
  - The geometric object to use display the data

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

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

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

- **inherit.aes**
  - If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders()`.
Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

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

**Aesthetics**

`geom_regon` understand the following aesthetics (required aesthetics are in bold):

- `x0`
- `y0`
- `sides`
- `r`
- `angle`
- `color`
- `fill`
- `size`
- `linetype`
- `alpha`
- `lineend`

**Computed variables**

`x`, `y` The coordinates for the corners of the polygon

**Examples**

```r
ggplot() +
  geom_regon(aes(x0 = runif(8), y0 = runif(8), sides = sample(3:10, 8),
               angle = 0, r = runif(8) / 10)) +
  coord_fixed()

# The polygons are drawn with `geom_shape`, so can be manipulated as such
ggplot() +
  geom_regon(aes(x0 = runif(8), y0 = runif(8), sides = sample(3:10, 8),
               angle = 0, r = runif(8) / 10),
             expand = unit(1, 'cm'), radius = unit(1, 'cm')) +
  coord_fixed()
```
**geom_shape**  
*Draw polygons with expansion/contraction and/or rounded corners*

**Description**

This geom is a cousin of `ggplot2::geom_polygon()` with the added possibility of expanding or contracting the polygon by an absolute amount (e.g. 1 cm). Furthermore, it is possible to round the corners of the polygon, again by an absolute amount. The resulting geom reacts to resizing of the plot, so the expansion/contraction and corner radius will not get distorted. If no expansion/contraction or corner radius is specified, the geom falls back to `geom_polygon` so there is no performance penalty in using this instead of `geom_polygon`.

**Usage**

```r
geom_shape(mapping = NULL, data = NULL, stat = "identity", position = "identity", expand = 0, radius = 0, ..., na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```

**Arguments**

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

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

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

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

- `expand` A numeric or unit vector of length one, specifying the expansion amount. Negative values will result in contraction instead. If the value is given as a numeric it will be understood as a proportion of the plot area width.

- `radius` As `expand` but specifying the corner radius.

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

- `na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.
show.legend logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

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

Aesthetics

gem_shape understand the following aesthetics (required aesthetics are in bold):

• x
• y
• color
• fill
• group
• size
• linetype
• alpha

Note

Some settings can result in the dissappearance of polygons, specifically when contracting or rounding corners with a relatively large amount. Also note that x and y scale limits does not take expansion into account and the resulting polygon might thus not fit into the plot.

Author(s)

Thomas Lin Pedersen

Examples

shape <- data.frame(
  x = c(0.5, 1, 0.75, 0.25, 0),
  y = c(0, 0.5, 1, 0.75, 0.25)
)
# Expand and round
ggplot(shape, aes(x = x, y = y)) +
  geom_shape(expand = unit(1, 'cm'), radius = unit(0.5, 'cm')) +
  geom_polygon(fill = 'red')

# Contract
ggplot(shape, aes(x = x, y = y)) +
  geom_polygon(fill = 'red') +
  geom_shape(expand = unit(-1, 'cm'))

# Only round corners
ggplot(shape, aes(x = x, y = y)) +
  geom_polygon(fill = 'red') +
  geom_shape(radius = unit(1, 'cm'))
Description

The sina plot is a data visualization chart suitable for plotting any single variable in a multiclass dataset. It is an enhanced jitter strip chart, where the width of the jitter is controlled by the density distribution of the data within each class.

Usage

```r
stat_sina(mapping = NULL, data = NULL, geom = "sina",
position = "dodge", scale = "area", method = "density",
bw = "nrd0", kernel = "gaussian", maxwidth = NULL, adjust = 1,
bin_limit = 1, binwidth = NULL, bins = NULL, seed = NA, ...,
na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```

```r
gem_sina(mapping = NULL, data = NULL, stat = "sina",
position = "dodge", ..., na.rm = FALSE, show.legend = NA,
inherit.aes = TRUE)
```

Arguments

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

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

- **geom**: The geometric object to use display the data

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

- **scale**: How should each sina be scaled. Corresponds to the scale parameter in `ggplot2::geom_violin()`? Available are:
  - 'area' for scaling by the largest density/bin among the different sinas
  - 'count' as above, but in addition scales by the maximum number of points in the different sinas.
  - 'width' Only scale according to the `maxwidth` parameter
For backwards compatibility it can also be a logical with TRUE meaning area and FALSE meaning width.

**method**

Choose the method to spread the samples within the same bin along the x-axis. Available methods: "density", "counts" (can be abbreviated, e.g. "d"). See Details.

**bw**

the smoothing bandwidth to be used. The kernels are scaled such that this is the standard deviation of the smoothing kernel. (Note this differs from the reference books cited below, and from S-PLUS.)

bw can also be a character string giving a rule to choose the bandwidth. See `bw.nrd`.

The default, "nrd0", has remained the default for historical and compatibility reasons, rather than as a general recommendation, where e.g., "SJ" would rather fit, see also Venables and Ripley (2002).

The specified (or computed) value of bw is multiplied by adjust.

**kernel**

a character string giving the smoothing kernel to be used. This must partially match one of "gaussian", "rectangular", "triangular", "epanechnikov", "biweight", "cosine" or "optcosine", with default "gaussian", and may be abbreviated to a unique prefix (single letter).

"cosine" is smoother than "optcosine", which is the usual ‘cosine’ kernel in the literature and almost MSE-efficient. However, "cosine" is the version used by S.

**maxwidth**

Control the maximum width the points can spread into. Values between 0 and 1.

**adjust**

the bandwidth used is actually adjust*bw. This makes it easy to specify values like ‘half the default’ bandwidth.

**bin_limit**

If the samples within the same y-axis bin are more than bin_limit, the samples’s X coordinates will be adjusted.

**binwidth**

The width of the bins. The default is to use bins bins that cover the range of the data. You should always override this value, exploring multiple widths to find the best to illustrate the stories in your data.

**bins**

Number of bins. Overridden by binwidth. Defaults to 50.

**seed**

A seed to set for the jitter to ensure a reproducible plot

**...**

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

**na.rm**

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

**show.legend**

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

**inherit.aes**

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

**stat**

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

There are two available ways to define the x-axis borders for the samples to spread within:

- method == "density"
  A density kernel is estimated along the y-axis for every sample group, and the samples are spread within that curve. In effect this means that points will be positioned randomly within a violin plot with the same parameters.
- method == "counts"
  The borders are defined by the number of samples that occupy the same bin.

Aesthetics

geom_sina understand the following aesthetics (required aesthetics are in bold):

- x
- y
- color
- group
- size
- alpha

Computed variables

- density  The density or sample counts per bin for each point
- scaled  density scaled by the maximum density in each group
- n  The number of points in the group the point belong to

Author(s)

Nikos Sidiropoulos, Claus Wilke, and Thomas Lin Pedersen

Examples

```r
ggplot(midwest, aes(state, area)) + geom_point()

# Boxplot and Violin plots convey information on the distribution but not the number of samples, while Jitter does the opposite.
   ggplot(midwest, aes(state, area)) + geom_violin()

   ggplot(midwest, aes(state, area)) + geom_jitter()

# Sina does both!
   ggplot(midwest, aes(state, area)) + geom_violin() + geom_sina()
```
p <- ggplot(midwest, aes(state, popdensity)) +
  scale_y_log10()

p + geom_sina()

# Colour the points based on the data set's columns
p + geom_sina(aes(colour = inmetro))

# Or any other way
cols <- midwest$popdensity > 10000
p + geom_sina(colour = cols + 1L)

# Sina plots with continuous x:
pp <- ggplot(midwest, aes(cut_width(area, 0.02), popdensity)) +
  geom_sina() +
  scale_y_log10()

### Sample gaussian distributions
# Unimodal
a <- rnorm(500, 6, 1)
b <- rnorm(400, 5, 1.5)

# Bimodal
c <- c(rnorm(200, 3, .7), rnorm(50, 7, 0.4))

# Trimodal
d <- c(rnorm(200, 2, 0.7), rnorm(300, 5.5, 0.4), rnorm(100, 8, 0.4))

df <- data.frame(
  'Distribution' = c('Unimodal 1', length(a)),
  'Value' = c(a, b, c, d),
)

df$Distribution <- factor(
  df$Distribution,
  levels(df$Distribution)[c(3, 4, 1, 2)]
)

p <- ggplot(df, aes(Distribution, Value))
p + geom_boxplot()
p + geom_violin() + geom_sina()

# By default, Sina plot scales the width of the class according to the width
# of the class with the highest density. Turn group-wise scaling off with:
geom_violin() +
geom_sina(scale = FALSE)

draw_spiro

geom_spiro

Draw spirograms based on the radii of the different "wheels" involved

Description

This, rather pointless, geom allows you to draw spirograms, as known from the popular drawing toy where lines were traced by inserting a pencil into a hole in a small gear that would then trace around inside another gear. The potential practicality of this geom is slim and it exists mainly for fun and art.

Usage

stat_spiro(mapping = NULL, data = NULL, geom = "path",
position = "identity", na.rm = FALSE, n = 500,
revolutions = NULL, show.legend = NA, inherit.aes = TRUE, ...)

geom_spiro(mapping = NULL, data = NULL, stat = "spiro",
position = "identity", arrow = NULL, n = 500, lineend = "butt",
na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)

Arguments

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

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

geom The geometric object to use display the data

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

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

n The number of points that should be used to draw a fully closed spirogram. If revolutions < 1 the actual number of points will be less than this.
revolutions  The number of times the inner gear should revolve around inside the outer gear. If NULL the number of revolutions to reach the starting position is calculated and used.

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().

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

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

arrow  Arrow specification, as created by grid::arrow().

lineend  Line end style (round, butt, square).

Aesthetics

stat.spiro and geom.spiro understand the following aesthetics (required aesthetics are in bold):

• R
• r
• d
• x0
• y0
• outer
• color
• size
• linetype
• alpha

Computed variables

x, y  The coordinates for the path describing the spirogram

index  The progression along the spirogram mapped between 0 and 1

Examples

# Basic usage
ggplot() +
  geom_spiro(aes(R = 10, r = 3, d = 5))

# Only draw a portion
ggplot() +
  geom_spiro(aes(R = 10, r = 3, d = 5), revolutions = 1.2)
# Let the inner gear circle the outside of the outer gear

ggplot() +
  geom_spiro(aes(R = 10, r = 3, d = 5, outer = TRUE))

Voronoi tessellation and delaunay triangulation

Description

This set of geoms and stats allows you to display voronoi tessellation and delaunay triangulation, both as polygons and as line segments. Furthermore it lets you augment your point data with related summary statistics. The computations are based on the deldir::deldir() package.

Usage

gem_voronoi_tile(mapping = NULL, data = NULL, stat = "voronoi_tile",
  position = "identity", na.rm = FALSE, bound = NULL, eps = 1e-09,
  max.radius = NULL, normalize = FALSE, asp.ratio = 1, expand = 0,
  radius = 0, show.legend = NA, inherit.aes = TRUE, ...)

gem_voronoi_segment(mapping = NULL, data = NULL,
  stat = "voronoi_segment", position = "identity", na.rm = FALSE,
  bound = NULL, eps = 1e-09, normalize = FALSE, asp.ratio = 1,
  show.legend = NA, inherit.aes = TRUE, ...)

gem_delaunay_tile(mapping = NULL, data = NULL,
  stat = "delaunay_tile", position = "identity", na.rm = FALSE,
  bound = NULL, eps = 1e-09, normalize = FALSE, asp.ratio = 1,
  expand = 0, radius = 0, show.legend = NA, inherit.aes = TRUE,
  ...)

gem_delaunay_segment(mapping = NULL, data = NULL,
  stat = "delaunay_segment", position = "identity", na.rm = FALSE,
  bound = NULL, eps = 1e-09, normalize = FALSE, asp.ratio = 1,
  expand = 0, show.legend = NA, inherit.aes = TRUE, ...)

gem_delaunay_segment2(mapping = NULL, data = NULL,
  stat = "delaunay_segment2", position = "identity", na.rm = FALSE,
  bound = NULL, eps = 1e-09, normalize = FALSE, asp.ratio = 1,
  n = 100, show.legend = NA, inherit.aes = TRUE, ...)

stat_delvor_summary(mapping = NULL, data = NULL, geom = "point",
  position = "identity", na.rm = FALSE, bound = NULL, eps = 1e-09,
  normalize = FALSE, asp.ratio = asp.ratio, show.legend = NA,
  inherit.aes = TRUE, ...)
Arguments

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

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

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

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

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

bound
The bounding rectangle for the tesselation or a custom polygon to clip the tesselation to. Defaults to NULL which creates a rectangle expanded 10% in all directions. If supplied as a bounding box it should be a vector giving the bounds in the following order: xmin, xmax, ymin, ymax. If supplied as a polygon it should either be a 2-column matrix or a `data.frame` containing an x and y column.

eps
A value of epsilon used in testing whether a quantity is zero, mainly in the context of whether points are collinear. If anomalous errors arise, it is possible that these may averted by adjusting the value of eps upward or downward.

max.radius
The maximum distance a tile can extend from the point of origin. Will in effect clip each tile to a circle centered at the point with the given radius. If `normalize` = TRUE the radius will be given relative to the normalized values.

normalize
Should coordinates be normalized prior to calculations. If x and y are in wildly different ranges it can lead to tesselation and triangulation that seems off when plotted without `ggplot2::coord_fixed()`. Normalization of coordinates solves this. The coordinates are transformed back after calculations.

asp.ratio
If `normalize` = TRUE the x values will be multiplied by this amount after normalization.

expand
A numeric or unit vector of length one, specifying the expansion amount. Negative values will result in contraction instead. If the value is given as a numeric it will be understood as a proportion of the plot area width.

radius
As expand but specifying the corner radius.

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.
geom_voronoi

**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()`.

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

**n**  The number of points to create for each segment

**geom**  The geometric object to use display the data

### Aesthetics

`geom_voronoi_tile` and `geom_delaunay_tile` understand the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- **alpha**
- **color**
- **fill**
- **linetype**
- **size**

`geom_voronoi_segment`, `geom_delaunay_segment`, and `geom_delaunay_segment2` understand the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- **alpha**
- **color**
- **linetype**
- **size**

### Computed variables

`stat_delvor_summary` computes the following variables:

- **x, y**  If `switch.centroid = TRUE` this will be the coordinates for the voronoi tile centroid, otherwise it is the original point
- **xcent, ycent**  If `switch.centroid = FALSE` this will be the coordinates for the voronoi tile centroid, otherwise it will be NULL
- **xorig, yorig**  If `switch.centroid = TRUE` this will be the coordinates for the original point, otherwise it will be NULL
- **ntri**  Number of triangles emanating from the point
- **triarea**  The total area of triangles emanating from the point divided by 3
**geom_voronoi**

`triarea` divided by the sum of the area of all triangles

**nsides** Number of sides on the voronoi tile associated with the point

**nedges** Number of sides of the associated voronoi tile that is part of the bounding box

**vorarea** The area of the voronoi tile associated with the point

**vorprop** `vorarea` divided by the sum of all voronoi tiles

**Examples**

# Voronoi
# You usually wants all points to take part in the same tesselation so set
# the group aesthetic to a constant (-1L is just a convention)
ggplot(iris, aes(Sepal.Length, Sepal.Width, group = -1L)) +
  geom_voronoi_tile(aes(fill = Species)) +
  geom_voronoi_segment() +
  geom_text(aes(label = stat(nsides), size = stat(vorarea)),
            stat = 'delvor_summary', switch.centroid = TRUE
  )

# Difference of normalize = TRUE (segment layer is calculated without
# normalisation)
ggplot(iris, aes(Sepal.Length, Sepal.Width, group = -1L)) +
  geom_voronoi_tile(aes(fill = Species), normalize = TRUE) +
  geom_voronoi_segment()

# Set a max radius
ggplot(iris, aes(Sepal.Length, Sepal.Width, group = -1L)) +
  geom_voronoi_tile(aes(fill = Species), colour = 'black', max.radius = 0.25)

# Set custom bounding polygon
triangle <- cbind(c(3, 9, 6), c(1, 1, 6))
ggplot(iris, aes(Sepal.Length, Sepal.Width, group = -1L)) +
  geom_voronoi_tile(aes(fill = Species), colour = 'black', bound = triangle)

# Use geom_shape functionality to round corners etc
ggplot(iris, aes(Sepal.Length, Sepal.Width, group = -1L)) +
  geom_voronoi_tile(aes(fill = Species), colour = 'black',
                  expand = unit(-.5, 'mm'), radius = unit(2, 'mm'))

# Delaunay triangles
ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_delaunay_tile(alpha = 0.3, colour = 'black')

# Use geom_delaunay_segment2 to interpolate aesthetics between end points
ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_delaunay_segment2(aes(colour = Species, size = 2,
                             lineend = 'round')
linear_trans

Create a custom linear transformation

Description

This function lets you compose transformations based on a sequence of linear transformations. If the transformations are parameterised the parameters will become arguments in the transformation function. The transformations are one of rotate, shear, stretch, translate, and reflect.

Usage

linear_trans(...)

rotate(angle)

stretch(x, y)

shear(x, y)

translate(x, y)

reflect(x, y)

Arguments

... A number of transformation functions.
angle An angle in radians
x the transformation magnitude in the x-direction
y the transformation magnitude in the x-direction

Value

linear_trans creates a trans object. The other functions return a 3x3 transformation matrix.

Examples

trans <- linear_trans(rotate(a), shear(1, 0), translate(x1, y1))
square <- data.frame(x = c(0, 0, 1, 1), y = c(0, 1, 1, 0))
square2 <- trans$transform(square$x, square$y, a = pi / 3, x1 = 4, y1 = 8)
square3 <- trans$transform(square$x, square$y, a = pi / 1.5, x1 = 2, y1 = -6)
square <- rbind(square, square2, square3)
square$group <- rep(1:3, each = 4)
ggplot(square, aes(x, y, group = group)) +
  geom_polygon(aes(fill = factor(group)), colour = 'black')
n_pages

Determine the number of pages in a paginated facet plot

Description

This is a simple helper that returns the number of pages it takes to plot all panels when using `facet_wrap_paginate()` and `facet_grid_paginate()`. It partially builds the plot so depending on the complexity of your plot it might take some time to calculate...

Usage

n_pages(plot)

Arguments

plot A ggplot object using either facet_wrap_paginate or facet_grid_paginate

Value

If the plot uses using either facet_wrap_paginate or facet_grid_paginate it returns the total number of pages. Otherwise it returns NULL

Examples

```r
p <- ggplot(diamonds) + geom_point(aes(carat, price), alpha = 0.1) +
  facet_wrap_paginate(~ cut:clarity, ncol = 3, nrow = 3, page = 1)
n_pages(p)
```

position_auto

Jitter based on scale types

Description

This position adjustment is able to select a meaningful jitter of the data based on the combination of positional scale types. It behaves differently depending on if none, one, or both the x and y scales are discrete. If both are discrete it will jitter the datapoints evenly inside a disc, if one of them is discrete it will jitter the discrete dimension to follow the density along the other dimension (like a sina plot). If neither are discrete it will not do any jittering.

Usage

```r
position_auto(jitter.width = 0.75, bw = "nrd0", scale = TRUE, seed = NA)
```
position_jitternormal

Arguments

- **jitter.width**: The maximal width of the jitter.
- **bw**: The smoothing bandwidth to use in the case of sina jittering. See the bw argument in `stats::density`.
- **scale**: Should the width of jittering be scaled based on the number of points in the group.
- **seed**: A seed to supply to make the jittering reproducible across layers.

See Also

- `geom_autopoint` for a point geom that uses auto-position by default.

Examples

```r
# Continuous vs continuous: No jitter
ggplot(mpg) + geom_point(aes(cty, hwy), position = 'auto')

# Continuous vs discrete: sina jitter
ggplot(mpg) + geom_point(aes(cty, drv), position = 'auto')

# Discrete vs discrete: disc-jitter
ggplot(mpg) + geom_point(aes(fl, drv), position = 'auto')

# Don't scale the jitter based on group size
ggplot(mpg) + geom_point(aes(cty, drv), position = position_auto(scale = FALSE))
ggplot(mpg) + geom_point(aes(fl, drv), position = position_auto(scale = FALSE))
```

Description

`ggplot2::geom_jitter()` adds random noise to points using a uniform distribution. When many points are plotted, they appear in a rectangle. This position jitters points using a normal distribution instead, resulting in more circular clusters.

Usage

`position_jitternormal(sd_x = NULL, sd_y = NULL)`

Arguments

- **sd_x**, **sd_y**: Standard deviation to add along the x and y axes. The function uses `stats::rnorm()` with mean = 0 behind the scenes. If omitted, defaults to 0.15. As with `ggplot2::geom_jitter()`, categorical data is aligned on the integers, so a standard deviation of more than 0.2 will spread the data so it’s not possible to see the distinction between the categories.
Examples

# Example data
df <- data.frame(
    x = sample(1:3, 1500, TRUE),
    y = sample(1:3, 1500, TRUE)
)

# position_jitter results in rectangular clusters
ggplot(df, aes(x = x, y = y)) +
    geom_point(position = position_jitter())

# geom_jitternormal results in more circular clusters
ggplot(df, aes(x = x, y = y)) +
    geom_point(position = position_jitternormal())

# You can adjust the standard deviations along both axes
# Tighter circles
ggplot(df, aes(x = x, y = y)) +
    geom_point(position = position_jitternormal(sd_x = 0.08, sd_y = 0.08))

# Oblong shapes
ggplot(df, aes(x = x, y = y)) +
    geom_point(position = position_jitternormal(sd_x = 0.2, sd_y = 0.08))

# Only add random noise to one dimension
ggplot(df, aes(x = x, y = y)) +
    geom_point(
        position = position_jitternormal(sd_x = 0.15, sd_y = 0),
        alpha = 0.1
    )

power_trans(n)

Description

This function can be used to create a proper trans object that encapsulates a power transformation \((x^n)\).

Usage

power_trans(n)

Arguments

n

The degree of the power transformation

Value

A trans object
**Examples**

```r
# Power of 5 transformations
trans <- power_trans(2)
trans$transform(1:10)

# Cubic root transformation
trans <- power_trans(1 / 3)
trans$transform(1:10)

# Use it in a plot
ggplot() +
  geom_line(aes(x = 1:10, y = 1:10)) +
  scale_x_continuous(trans = power_trans(2),
                     expand = c(0, 1))
```

---

**Description**

This function creates a trans object that converts radial data to their corresponding coordinates in cartesian space. The trans object is created for a specific radius and angle range that will be mapped to the unit circle so data doesn’t have to be normalized to 0-1 and 0-2*pi in advance. While there exists a clear mapping from radial to cartesian, the inverse is not true as radial representation is periodic. It is impossible to know how many revolutions around the unit circle a point has taken from reading its coordinates. The inverse function will always assume that coordinates are in their first revolution i.e. map them back within the range of a.range.

**Usage**

```r
radial_trans(r.range, a.range, offset = pi/2, pad = 0.5,
             clip = FALSE)
```

**Arguments**

- `r.range`: The range in radius that corresponds to 0 - 1 in the unit circle.
- `a.range`: The range in angles that correspond to 2*pi - 0. As radians are normally measured counterclockwise while radial displays are read clockwise it’s an inverse mapping.
- `offset`: The offset in angles to apply. Determines that start position on the circle. pi/2 (the default) corresponds to 12 o’clock.
- `pad`: Adds to the end points of the angle range in order to separate the start and end point. Defaults to 0.5
- `clip`: Should input data be clipped to r.range and a.range or be allowed to extend beyond. Defaults to FALSE (no clipping).
Value

A trans object. The transform method for the object takes an r (radius) and a (angle) argument and returns a data.frame with x and y columns with rows for each element in r/a. The inverse method takes an x and y argument and returns a data.frame with r and a columns and rows for each element in x/y.

Note

While trans objects are often used to modify scales in ggplot2, radial transformation is different as it is a coordinate transformation and takes two arguments. Consider it a trans version of coord_polar and use it to transform your data prior to plotting.

Examples

```r
# Some data in radial form
rad <- data.frame(r = seq(1, 10, by = 0.1), a = seq(1, 10, by = 0.1))

# Create a transformation
radial <- radial_trans(c(0, 1), c(0, 5))

# Get data in x, y
cart <- radial$transform(rad$r, rad$a)

# Have a look
ggplot() +
  geom_path(aes(x = x, y = y), data = cart, color = 'forestgreen') +
  geom_path(aes(x = r, y = a), data = rad, color = 'firebrick')
```

scale_depth

Scales for depth perception

Description

These scales serve to scale the depth aesthetic when creating stereographic plots. The range specifies the relative distance between the points and the paper plane in relation to the distance between the eyes and the paper plane i.e. a range of c(-0.5, 0.5) would put the highest values midways between the eyes and the image plane and the lowest values the same distance behind the image plane. To ensure a nice viewing experience these values should not exceed ~0.3 as it would get hard for the eyes to consolidate the two pictures.

Usage

```r
scale_depth(..., range = c(0, 0.3))

scale_depth_continuous(..., range = c(0, 0.3))

scale_depth_discrete(..., range = c(0, 0.3))
```
Arguments

... arguments passed on to continuous_scale or discrete_scale

range The relative range as related to the distance between the eyes and the paper plane.

Examples

```r
ggplot(mtcars) +
  geom_point(aes(mpg, disp, depth = cyl)) +
  scale_depth(range = c(-0.1, 0.25)) +
  facet_stereo()
```

scale_unit Position scales for units data

Description

These are the default scales for the units class. These will usually be added automatically. To override manually, use scale_*_unit.

Usage

```r
scale_x_unit(name = waiver(), breaks = waiver(), unit = NULL, minor_breaks = waiver(), labels = waiver(), limits = NULL, expand = waiver(), oob = censor, na.value = NA_real_, trans = "identity", position = "bottom", sec.axis = waiver())
```

```r
scale_y_unit(name = waiver(), breaks = waiver(), unit = NULL, minor_breaks = waiver(), labels = waiver(), limits = NULL, expand = waiver(), oob = censor, na.value = NA_real_, trans = "identity", position = "left", sec.axis = waiver())
```

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

unit A unit specification to use for the axis. If given, the values will be converted to this unit before plotting. An error will be thrown if the specified unit is incompatible with the unit of the data.
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.

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)
• A function that takes the breaks as input and returns labels as output

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

expand 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 expand_scale() 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.

oob Function that handles limits outside of the scale limits (out of bounds). The default replaces out of bounds values with NA.

na.value Missing values will be replaced with this value.

trans Either 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 name_trans, e.g. scales::boxcox_trans(). You can create your own transformation with scales::trans_new().

position The position of the axis. "left" or "right" for vertical scales, "top" or "bottom" for horizontal scales

Examples
library(units)
gallon <- as_units('gallon')
mtcars$consumption <- mtcars$mpg * with(ud_units, mi / gallon)
mtcars$power <- mtcars$hp * with(ud_units, hp)
# Use units encoded into the data
ggplot(mtcars) +
  geom_point(aes(power, consumption))

# Convert units on the fly during plotting
ggplot(mtcars) +
  geom_point(aes(power, consumption)) +
  scale_x_unit(unit = 'W') +
  scale_y_unit(unit = 'km/l')

# Resolve units when transforming data
ggplot(mtcars) +
  geom_point(aes(power, 1 / consumption))

theme_no_axes  

---

**theme_no_axes**  

Theme without axes and gridlines

**Description**

This theme is a simple wrapper around any complete theme that removes the axis text, title and ticks as well as the grid lines for plots where these have little meaning.

**Usage**

```
theme_no_axes(base.theme = theme_bw())
```

**Arguments**

- `base.theme`  
  The theme to use as a base for the new theme. Defaults to `ggplot2::theme_bw()`.

**Value**

A modified version of `base.theme`

**Examples**

```
p <- ggplot() + geom_point(aes(x = wt, y = qsec), data = mtcars)
p + theme_no_axes()
p + theme_no_axes(theme_grey())
```
trans_reverser  

Reverse a transformation

Description

While the scales package export a reverse_trans object it does not allow for reversing of already transformed ranged - e.g. a reverse exp transformation is not possible. trans_reverser takes a trans object or something coercible to one and creates a reverse version of it.

Usage

trans_reverser(trans)

Arguments

trans  
A trans object or an object that can be converted to one using scales::as.trans()

Value

A trans object

Examples

# Lets make a plot
p <- ggplot() +
  geom_line(aes(x = 1:10, y = 1:10))

# scales already have a reverse trans
p + scale_x_continuous(trans = 'reverse')

# But what if you wanted to reverse an already log transformed scale?
p + scale_x_continuous(trans = trans_reverser('log'))
# Index

*Topic datasets

  GeomShape, 14

aes(), 15, 18, 21, 22, 24, 27, 30, 32, 35, 37, 39, 42, 44, 48, 51, 55, 59, 61, 63, 65, 69, 72

aes_(), 15, 18, 21, 22, 24, 27, 30, 32, 35, 37, 39, 42, 44, 48, 51, 55, 59, 61, 63, 65, 69, 72

borders(), 15, 18, 21, 23, 25, 28, 30, 33, 35, 38, 40, 42, 45, 49, 53, 56, 59, 61, 64, 66, 70, 73

bw.nrd, 66

deldir::deldir(), 71
density(), 21
diagonal, 37

eexpand_scale(), 82

facet_col(facet_row), 8
facet_grid_paginate, 4, 10, 12, 13
facet_grid_paginate(), 76
facet_matrix, 6, 22, 23
facet_matrix(), 20
facet_row, 8
facet_stereo, 5, 9, 12, 13
facet_wrap_paginate, 5, 10, 10, 13
facet_wrap_paginate(), 76
facet_zoom, 5, 10, 12, 12
FacetCol (GeomShape), 14
FacetGridPaginate (GeomShape), 14
FacetMatrix (GeomShape), 14
FacetRow (GeomShape), 14
FacetStereo (GeomShape), 14
FacetWrapPaginate (GeomShape), 14
FacetZoom (GeomShape), 14

fortify(), 15, 18, 21, 22, 24, 27, 30, 32, 35, 37, 39, 42, 44, 48, 51, 55, 59, 61, 63, 65, 69, 72

gather_set_data, 13
gather_set_data(), 60
groupArc, 14
groupArc0 (geom_arc), 14
groupArc2 (geom_arc), 14
groupArc_bar, 17
groupArc_bar(), 16, 33
groupAuto-density, 7, 20
groupAutoHistogram, 7
groupAutoHistogram (geom_auto-density), 20
groupAutoPoint, 7, 22, 77
groupBezier, 23
groupBezier(), 27, 34
groupBezier0 (geomBezier), 23
groupBezier2 (geomBezier), 23
groupB spline, 26
groupBspline0 (geomBspline), 26
groupBspline2 (geomBspline), 26
groupBspline_closed, 29
groupBspline_closed0
  (geomBspline_closed), 29
groupCircle, 32
groupCircle(), 39
groupDelaunay (geomVoronoi), 71
groupDelaunay_segment (geomVoronoi), 71
groupDelaunay_segment2 (geomVoronoi), 71
groupDelaunay_tile (geomVoronoi), 71
groupDiagonal, 34
groupDiagonal0 (geomDiagonal), 34
groupDiagonal2 (geomDiagonal), 34
groupDiagonal wide, 37
groupEllipse, 39
groupLink, 41
groupLink(), 24
groupLink0 (geomLink), 41
groupLink2 (geomLink), 41
geom_link2(), 24
geom_mark_circle, 44, 50, 54, 57
geom_mark_ellipse, 46, 47, 54, 57
geom_mark_hull, 46, 50, 51, 57
geom_mark_rect, 46, 50, 54, 55
geom_parallel_sets, 58
geom_parallel_sets_axes
  (geom_parallel_sets), 58
geom_parallel_sets_labels
  (geom_parallel_sets), 58
geom_regon, 61
geom_shape, 63
geom_shape(), 44
geom_sina, 65
geom_spiro, 69
ggplot2::geom_point(), 6, 22, 32, 33
geom_voronoi_segment (geom_voronoi), 71
ggplot2::geom_polygon(), 63
ggplot2::geom_segment(), 41
geom_voronoi_tile (geom_voronoi), 71
ggplot2::geom_voronoi(), 44
GeomArc (GeomShape), 14
GeomArc0 (GeomShape), 14
GeomArcBar (GeomShape), 14
GeomAutoarea (GeomShape), 14
GeomAutoRect (GeomShape), 14
GeomBezier0 (GeomShape), 14
GeomBspline0 (GeomShape), 14
GeomBsplineClosed0 (GeomShape), 14
GeomCircle (GeomShape), 14
GeomMarkCircle (GeomShape), 14
GeomMarkEllipse (GeomShape), 14
GeomMarkHull (GeomShape), 14
GeomMarkRect (GeomShape), 14
GeomParallelSetsAxes (GeomShape), 14
GeomPathInterpolate (GeomShape), 14
GeomShape, 14
ggforce (ggforce-package), 3
ggforce-extensions (GeomShape), 14
ggforce-package, 3
ggplot(), 15, 18, 21, 22, 24, 27, 30, 32, 35,
  37, 39, 42, 44, 48, 51, 55, 59, 61, 63,
  65, 69, 72
ggplot2::coord_fixed(), 32, 72
ggplot2::coord_polar(), 14
ggplot2::facet_grid(), 4, 8
ggplot2::facet_wrap(), 8, 10
ggplot2::geom_density(), 20
ggplot2::geom_histogram(), 20
ggplot2::geom_jitter(), 77
ggplot2::geom_path(), 41
labeller(), 5, 9, 11
layer(), 15, 18, 21, 23, 25, 28, 30, 33, 35, 38,
  40, 42, 45, 49, 53, 56, 60, 62, 63, 66,
  70, 73
linear_trans, 75
n_pages(), 76
n_pages(), 5, 12
position_auto, 7, 23, 76
position_auto(), 22
position_jitternormal, 77
PositionAuto (GeomShape), 14
PositionFloatstack (GeomShape), 14
PositionJitterNormal (GeomShape), 14
power_trans, 78
radial_trans, 79
reflect (linear_trans), 75
rotate (linear_trans), 75
scale_depth, 80
scale_depth(), 10
scale_depth_continuous (scale_depth), 80
scale_depth_discrete (scale_depth), 80
scale_type.units (scale_unit), 81
scale_unit, 81
scale_x_unit (scale_unit), 81
scale_y_unit (scale_unit), 81
ScaleContinuousPositionUnit
  (GeomShape), 14
scales::as.trans(), 84
scales::boxcox_trans(), 82
scales::trans_new(), 82
shear (linear_trans), 75
stat_arc (geom_arc), 14
INDEX

stat_arc0 (geom_arc), 14
stat_arc2 (geom_arc), 14
stat_arc_bar (geom_arc_bar), 17
stat_bezier (geom_bezier), 23
stat_bezier0 (geom_bezier), 23
stat_bezier2 (geom_bezier), 23
stat_bspline (geom_bspline), 26
stat_bspline0 (geom_bspline), 26
stat_bspline2 (geom_bspline), 26
stat_bspline_closed (geom_bspline_closed), 29
stat_circle (geom_circle), 32
stat_delvor_summary (geom_voronoi), 71
stat_diagonal (geom_diagonal), 34
stat_diagonal0 (geom_diagonal), 34
stat_diagonal2 (geom_diagonal), 34
stat_diagonal_wide (geom_diagonal_wide), 37
stat_ellip (geom_ellipse), 39
stat_link (geom_link), 41
stat_link2 (geom_link), 41
stat_parallel_sets (geom_parallel_sets), 58
stat_parallel_sets_axes (geom_parallel_sets), 58
stat_pie (geom_arc_bar), 17
stat_regon (geom_region), 61
stat_sina (geom_sina), 65
stat_spiro (geom_spiro), 69
StatArc (GeomShape), 14
StatArc0 (GeomShape), 14
StatArc2 (GeomShape), 14
StatArcBar (GeomShape), 14
StatAutobin (GeomShape), 14
StatAutodensity (GeomShape), 14
StatBezier (GeomShape), 14
StatBezier0 (GeomShape), 14
StatBezier2 (GeomShape), 14
StatBspline (GeomShape), 14
StatBspline2 (GeomShape), 14
StatCircle (GeomShape), 14
StatDelaunaySegment (GeomShape), 14
StatDelaunaySegment2 (GeomShape), 14
StatDelaunayTile (GeomShape), 14
StatDelvorSummary (GeomShape), 14
StatDiagonal (GeomShape), 14
StatDiagonal0 (GeomShape), 14
StatDiagonal2 (GeomShape), 14

...