Geoms

`geom_abline`(aes(intercept, slope), alpha, colour, linetype, size)
The abline geom adds a line with specified slope and intercept to the plot.

`geom_area`(aes(x, ymin=0, ymax), alpha, colour, fill, linetype, size)
An area plot is the continuous analog of a stacked bar chart (see `geom_bar`), and can be used to show how composition of the whole varies over the range of x. Choosing the order in which different components is stacked is very important, as it becomes increasing hard to see the individual pattern as you move up the stack.

`geom_bar`(aes(x), alpha, colour, fill, linetype, size, weight)
The bar geom is used to produce 1d area plots: bar charts for categorical x, and histograms for continuous y. `stat_bin` explains the details of these summaries in more detail. In particular, you can use the weight aesthetic to create weighted histograms and barcharts where the height of the bar no longer represent a count of observations, but a sum over some other variable.

`geom_bin2d`(aes(xmax, xmin, ymax, ymin), alpha, colour, fill, linetype, size, weight)
Add heatmap of 2d bin counts.

`geom_blank`(aes(),)
The blank geom draws nothing, but can be a useful way of ensuring common scales between different plots.

`geom_boxplot`(aes(lower, middle, upper, x, ymax, ymin), alpha, colour, fill, linetype, shape, size, weight, notch=FALSE,)
The upper and lower "hinges" correspond to the first and third quartiles (the 25th and 75th percentiles). This differs slightly from the method used by the `boxplot` function, and may be apparent with small samples. See `boxplot.stats` for for more information on how hinge positions are calculated for `boxplot`.

`geom_contour`(aes(x, y), alpha, colour, linetype, size, weight)
Display contours of a 3d surface in 2d. See `stat_contour`.

`geom_crossbar`(aes(x, y, ymax, ymin), alpha, colour, fill, linetype, size)
Hollow bar with middle indicated by horizontal line. See `geom_linerange`.

`geom_density`(aes(x, y), alpha, colour, fill, linetype, size, weight)
A smooth density estimate calculated by `stat_density`.

`geom_density2d`(aes(x, y), alpha, colour, fill, linetype, size)
Perform a 2D kernel density estimatation using `kde2d` and display the results with contours.

`geom_dotplot`(aes(x, y), alpha, colour, fill)
In a dot plot, the width of a dot corresponds to the bin width (or maximum width, depending on the binning algorithm), and dots are stacked, with each dot representing one observation.

`geom_errorbar`(aes(x, ymax, ymin), alpha, colour, linetype, size, width)
Error bars.

`geom_freqpoly`(aes(x), alpha, colour, linetype, size)
Frequency polygon.

`geom_hex`(aes(x, y), alpha, colour, fill, size)
Hexagon binning.

`geom_histogram`(aes(x), alpha, colour, fill, linetype, size, weight)
`geom_histogram` is an alias for `geom_bar` plus `stat_bin` (look there to see parameters).

`geom_hline`(aes(yintercept), alpha, colour, linetype, size)
This geom allows you to annotate the plot with horizontal lines.

`geom_jitter`(aes(x, y), alpha, colour, fill, shape, size)
The jitter geom is a convenient default for `geom_point` with `position = 'jitter'`. See `position_jitter` to see how to adjust amount of jittering.

`geom_line`(aes(x, y), alpha, colour, linetype, size)
Connect observations, ordered by x value.

`geom_linerange`(aes(x, ymin, ymax),)
An interval represented by a vertical line

`geom_map`(aes(`map_id`), alpha, colour, fill, linetype, size)
Need data frame with map coordinates, with columns `x` or `long`, `y` or `lat`, and `region` or `id`. With `geom_polygon` will need two data frames - coordinates of the polygon (positions) and values for each polygon (values) linked by an id variable. `expand_limits`() may also be helpful.

`geom_path`(aes(x, y), alpha, colour, linetype, size)
Connect observations in original order.

`geom_point`(aes(x, y), alpha, colour, fill, shape, size)
Used to create scatterplots.

`geom_pointrange`(aes(x, y, ymax, ymin), alpha, colour, fill, linetype, shape, size)
An interval represented by a vertical line with a point in the middle. See `geom_linerange`.

`geom_polygon`(aes(x, y), alpha, colour, fill, linetype, size)
Polygon, a filled path.

`geom_quantile`(aes(x, y), alpha, colour, linetype, size, weight)
A continuous analogue of `geom_boxplot`.

`geom_raster`(aes(x, y), alpha, fill)
This is a special case of `geom_tile` where all tiles are the same size. It is implemented highly efficiently using the internal `rasterGrob` function.

`geom_rect`(aes(xmax, xmin, ymax, ymin), alpha, colour, fill, linetype, size)
2d rectangles.

`geom_ribbon`(aes(x, ymax, ymin), alpha, colour, fill, linetype, size)
Ribbons, y range with continuous x values.

`geom_rug`(aes(), alpha, colour, linetype, size)
Marginal rug plots.

`geom_segment`(aes(x, xend, y, yend), alpha, colour, linetype, size)
Single line segments.

`geom_smooth`(aes(x, y), alpha, colour, fill, linetype, size, weight)
Add a smoothed conditional mean. See `stat_smooth()`

`geom_step`(aes(x, y), alpha, colour, linetype, size)
Connect observations by stairs.

`geom_text`(aes(label, x, y), alpha, angle, colour, family, fontface, hjust, lineheight, size, vjust)
Textual annotations.

`geom_tile`(aes(x, y), alpha, colour, fill, linetype, size)
Similar to `levelplot` and `image`.

`geom_violin`(aes(x, y), alpha, colour, fill, linetype, size, weight)
Violin plot

`geom_vline`(aes(xintercept), alpha, colour, linetype, size)
This geom allows you to annotate the plot with vertical lines (see `geom_hline` and `geom_abline` for other types of lines.

Positions

`position_dodge`(width = NULL, height = NULL)
Adjust position by dodging overlaps to the side.

`position_fill`(width = NULL, height = NULL)
Stack overlapping objects on top of one another, and standardise to have equal height.

`position_identity`(width = NULL, height = NULL)
Don't adjust position

`position_stack`(width = NULL, height = NULL)
Stack overlapping objects on top of one another

`position_jitter`(width=NULL, height=NULL)
Jitter points to avoid overplotting.

Statistics

`stat_bin`(binwidth, breaks, origin, width, right=TRUE, drop=FALSE, ...)

`stat_bin2d`(bins, drop=FALSE, ...)

`stat_bindot`(binaxis="x", method="dotdensity", binwidth, binpositions, origin, right=TRUE, drop=FALSE, na.rm=FALSE, aes(), geom, position)

`stat_binhex`(bins=c(30, 30), na.rm=FALSE, ...)
Bin 2d plane into hexagons

`stat_boxplot`(coef=1.5, na.rm=FALSE, ...)
Calculates components of box and whisker plot.

`stat_contour`(na.rm=FALSE, ..., bins, binwidth)
Calculates contours of 3d data; bins gives number of contours, binwidth specifies the same thing by contour width. Also possible to map size or color to contour level by =..level..

`stat_density`(adjust, kernel, trim=TRUE, na.rm=FALSE, ...)
1d kernel density estimate.

`stat_density2d`(contour=TRUE, n, kde2d(...), na.rm=TRUE, ...)
2d density estimation. `kde2d(...)` is for other arguments to be passed to `kde2d`.

`stat_ecdf`(n, ...)
Empirical CDF of x. If n is NULL, do not interpolate, otherwise, interpolate over n points.

`stat_function`(fun, n, args, ...)
Superimpose a function, `fun`, n points to interpolate along, with `args()` to pass to `fun`.

`stat_identity`(width, height)
Identity statistic - width and height describe the width and height of the tiles.

`stat_qq`(distribution, dparams, ..., na.rm=FALSE, ...)
Calculation for quantile-quantile plot. distribution function `dist` with parameters `dparams` and other arguments ...

`stat_quantiles`(quantiles, formula, method="rq", na.rm=FALSE, ...)
quantiles of y to calculate, using `formula` and currently only supports method `rq`

`stat_smooth`(method, formula, se=TRUE, fullrange, level=0.95, n, na.rm=FALSE, ...)
Uses a smoother fit by one of `lm`, `glm`, `gam`, `loess`, or `rlm`.

`stat_spoke`(...)
convert angle and radius to xend and yend. Requires aes(angle, radius, x, y).

`stat_summary_hex`(bins, drop=TRUE, fun, ..., ...)
Apply function for 2d hexagonal bins. Bins from `stat_binhex` with `fun` for summary applied to each bin. ... includes function arguments as well as standard stat arguments

`stat_summary2d`(bins, drop, fun, ..., ...)
Apply function for 2d rectangular bins. Bins from `stat_bin2d` with `fun` for summary applied to each bin. ... includes function arguments as well as standard stat arguments

`stat_unique`(...)
Removes duplicates

`stat_ydensity`(trim=TRUE, scale="area", na.rm=FALSE, ..., adjust, kernel, ...)
1d kernel density estimate along y axis for violin plot. If scale="count" areas are scaled proportionate to the number of observations. If scale="width", all violins have the same maximum width.

`stat_sum`(...)
Sum unique values - useful for overplotting on scatterplots.

`stat_summary`(...)
Allows flexibility in specification of summary functions - either operating on the data frame with argument name `fun.data` or on a vector `fun.y, fun.ymax, fun.ymin`.

Coordinate systems

`coord_cartesian`(xlim, ylim)
Setting limits on the coordinate system will zoom the plot (like you're looking at it with a magnifying glass), and will not change the underlying data like setting limits on a scale will.

`coord_fixed`(ratio = 1, xlim = NULL, ylim = NULL)
Forces a specified ratio between the physical representation of data units on the axes. The ratio represents the number of units on the y-axis equivalent to one unit on the x-axis.

`coord_flip`(...)
Flipped cartesian coordinates so horizontal becomes vertical.

`coord_map`(projection = "mercator", ..., orientation = c(90, 0, mean(range(x))), xlim = NULL, ylim = NULL)
This coordinate system provides the full range of map projections available in the mapproj package. Alternate projections can be found in that package.

`coord_polar`(theta = "x", start = 0, direction = 1)
The polar coordinate system is most commonly used for pie charts, which are a stacked bar chart in polar coordinates.

`coord_trans`(xtrans = "identity", ytrans = "identity", limx = NULL, limy = NULL)
Different from scale transformations in that it occurs after statistical transformation and will affect the visual appearance of geoms - there is no guarantee that straight lines will continue to be straight. Currently works only with cts values.

Faceting

`facet_grid`(facets, margins = FALSE, scales = "fixed", space = "fixed", shrink = TRUE, labeller = "label_value", as.table = TRUE, drop = TRUE)
Lay out panels in a grid.

`facet_null`(shrink=TRUE)
Specifies a single panel. If shrink=TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.

`facet_wrap`(facets, nrow = NULL, ncol = NULL, scales = "fixed", shrink = TRUE, as.table = TRUE, drop = TRUE)
Wrap a 1d ribbon of panels into 2d.

`label_both`
Passed in `facet_grid` to the labeller argument. Labels with variable name and value.

`label_bquote`(...)
Passed in `facet_grid` to the labeller argument. See bquote for details on the syntax of the argument. The label value is x. Useful for facet labels that are expressions

`label_parsed`(...)
Passed in `facet_grid` to the labeller argument. Label facets with parsed label. Useful for facet labels that are expressions.

`label_value`(...)
Passed in `facet_grid` to the labeller argument. Default labels.

Scales

`expand_limits`(...)
named list of aesthetics specifying the value that should be included in each scale

`guides`(...)
List of scale guide pairs

`guide_legend`(title = waiver(), title.position = NULL, title.theme = NULL, title.hjust = NULL, title.vjust = NULL, label = TRUE, label.position = NULL, label.theme = NULL, label.hjust = NULL, label.vjust = NULL, keywidth = NULL, keyheight = NULL, direction = NULL, default.unit = "line", override.aes = list(), nrow = NULL, ncol = NULL, byrow = FALSE, reverse = FALSE, order = 0, ...)
Legend type guide shows key (i.e., geoms) mapped onto values. Legend guides for various scales are integrated if possible.

`guide_colorbar`(title = waiver(), title.position = NULL, title.theme = NULL, title.hjust = NULL, title.vjust = NULL, label = TRUE, label.position = NULL, label.theme = NULL, label.hjust = NULL, label.vjust = NULL, barwidth = NULL, barheight = NULL, nbin = 20, raster = TRUE, ticks = TRUE, draw.ulim = TRUE, draw.llim = TRUE, direction = NULL, default.unit = "line", reverse = FALSE, order = 0, ...)
Colour bar guide shows continuous color scales mapped onto values. Colour bar is available with `scale_fill` and `scale_colour`.

`scale_alpha`(..., range = c(0.1, 1))

`scale_area`(..., range=c(1,6))

`scale_colour_brewer`(..., type="seq", palette=1)
Substitute color or fill for colour. If palette is a string, will use that name, otherwise, will index the list of palettes.

`scale_colour_gradient`(..., low = "#132B43", high = "#56B1F7", space = "Lab", na.value = "grey50", guide = "colourbar")
Substitute color or fill for colour. Also aliases `scale_colour_continuous`.

`scale_colour_gradient2`(..., low = muted("red"), mid = "white", high = muted("blue"), midpoint = 0, space = "rgb", na.value = "grey50", guide = "colourbar")
Diverging color scheme. Substitute color or fill for colour.

`scale_colour_gradientn`(..., colours, values = NULL, space = "Lab", na.value = "grey50", guide = "colourbar")
Smooth color gradient between n colors. Substitute color or fill for colour.

`scale_colour_grey`(..., start = 0.2, end = 0.8, na.value = "red")

`scale_colour_hue`(..., h = c(0, 360) + 15, c = 100, l = 65, h.start = 0, direction = 1, na.value = "grey50")
Qualitative colour scale with evenly spaced hues. Substitute color or fill for colour. Also aliases `scale_colour_discrete`.

`scale_colour_identity`(..., guide="none")
Use values without scaling. Substitute fill, shape, linetype, alpha, size, color for colour.

`scale_colour_manual`(..., values)
Create your own discrete scale

`scale_linetype_discrete`(..., na.value = "blank")
Must be a discrete scale.

`scale_shape_discrete`(..., solid = TRUE)
Must be a discrete scale.

`scale_size`(..., range = c(1, 6))
Can be discrete (`scale_size_discrete`) or continuous. Range specifies minimum and maximum size of plotting symbols after transformation.

`scale_x_continuous`(..., expand=waiver())
Also works for y. Common parameters: name, breaks, labels, na.value, limits, trans. Aliases for transformations: `scale_x_log10`, `scale_x_reverse`, `scale_x_sqrt`.

`scale_x_date`(..., expand = waiver(), breaks = pretty_breaks(), minor_breaks = waiver())
Also works for y. Args: `breaks` = vector of breaks, `minor_breaks` = locations of minor breaks between labeled breaks.

`scale_x_datetime`(..., expand = waiver(), breaks = pretty_breaks(), minor_breaks = waiver())
Also works for y. Args: `breaks` = vector of breaks, `minor_breaks` = locations of minor breaks between labeled breaks.

`scale_x_discrete`(..., expand = waiver())
Also works for y. You can use continuous positions even with a discrete position scale - this allows you (e.g.) to place labels between bars in a bar chart. Continuous positions are numeric values starting at one for the first level, and increasing by one for each level (i.e. the labels are placed at integer positions). This is what allows jittering to work.

`labs`(title, x, y)

`xlab`(label)

`ggtitle`(title)

`update_labels`(p, labels)
p is the plot to modify, labels are a named list of new labels. Works for axis, legend labels.

`xlim`(...)
If numeric, will create a continuous scale, if factor or character, will create a discrete scale. Observations not in this range will be dropped completely and not passed to any other layers.

Themes

`add_theme`(t1, t2, t2name)
Modify properties of an element in a theme object. Add t1 to t2 and name it t2name.

`element_blank()`()
This theme element draws nothing, and assigns no space

`element_line`(colour = NULL, size = NULL, linetype = NULL, lineend = NULL, color = NULL)

`element_rect`(fill = NULL, colour = NULL, size = NULL, linetype = NULL, color = NULL)

`element_text`(family = NULL, face = NULL, colour = NULL, size = NULL, hjust = NULL, vjust = NULL, angle = NULL, lineheight = NULL, color = NULL)

`theme`(..., complete = FALSE)
Use this function to modify theme settings. Elements include line, rect, text, title, axis.title, axis.text, axis.ticks, axis.ticks.length, axis.ticks.margin, axis.line, legend.background, legend.box, panel.background, panel.border, panel.margin, panel.grid, plot.background, plot.title, plot.margin, strip.background, strip.text

`theme_bw`(base_size=12, base_family="")
A theme with white background and black gridlines.

`theme_grey`(base_size=12, base_family="")
A theme with grey background and white gridlines. (default theme)

`theme_classic`()
A classic-looking theme, with x and y axis lines and no gridlines.

`theme_minimal`()
A minimalistic theme with no background annotations.