Package ‘ggplot2’

December 30, 2016

Version 2.2.1

Title Create Elegant Data Visualisations Using the Grammar of Graphics

Description A system for 'declaratively' creating graphics, based on `The Grammar of Graphics`. You provide the data, tell 'ggplot2' how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.

Depends R (>= 3.1)

Imports digest, grid, gtable (>= 0.1.1), MASS, plyr (>= 1.7.1), reshape2, scales (>= 0.4.1), stats, tibble, lazyeval

Suggests covr, ggplot2movies, hexbin, Hmisc, lattice, mapproj, maps, maptools, mgcv, multcomp, nlme, testthat (>= 0.11.0), quantreg, rmarkdown, svglite

Enhances sp

License GPL-2 | file LICENSE


BugReports https://github.com/tidyverse/ggplot2/issues

LazyData true

Collate ‘ggproto.r’ ‘aaa-.r’ ‘aes-calculated.r’
‘aes-colour-fill-alpha.r’ ‘aes-group-order.r’
‘aes-linetype-size-shape.r’ ‘aes-position.r’ ‘utilities.r’
‘aes.r’ ‘legend-draw.r’ ‘geom-.r’ ‘annotation-custom.r’
‘annotation-logticks.r’ ‘geom-polygon.r’ ‘geom-map.r’
‘annotation-map.r’ ‘geom-raster.r’ ‘annotation-raster.r’
‘annotation.r’ ‘autoplot.r’ ‘axis-secondary.R’ ‘bench.r’
‘bin.R’ ‘coord-.r’ ‘coord-cartesian-.r’ ‘coord-fixed.r’
‘coord-flip.r’ ‘coord-map.r’ ‘coord-munch.r’ ‘coord-polar.r’
‘coord-quickmap.R’ ‘coord-transform.r’ ‘data.R’ ‘facet-.r’
‘facet-grid-.r’ ‘facet-null.r’ ‘facet-wrap.r’ ‘fortify-lm.r’
‘fortify-map.r’ ‘fortify-multcomp.r’ ‘fortify-spatial.r’
‘fortify.r’ ‘stat-.r’ ‘geom-abline.r’ ‘geom-rect.r’
‘geom-bar.r’ ‘geom-bin2d.r’ ‘geom-blank.r’ ‘geom-boxplot.r’
‘geom-col.r’ ‘geom-path.r’ ‘geom-contour.r’ ‘geom-count.r’
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>+.gg</td>
<td>5</td>
</tr>
<tr>
<td>aes</td>
<td>6</td>
</tr>
<tr>
<td>aes_</td>
<td>7</td>
</tr>
<tr>
<td>aes_colour_fill_alpha</td>
<td>8</td>
</tr>
<tr>
<td>aes_group_order</td>
<td>9</td>
</tr>
<tr>
<td>aes_linetype_size_shape</td>
<td>11</td>
</tr>
<tr>
<td>aes_position</td>
<td>12</td>
</tr>
<tr>
<td>annotate</td>
<td>13</td>
</tr>
<tr>
<td>annotation_custom</td>
<td>14</td>
</tr>
<tr>
<td>annotation_logticks</td>
<td>15</td>
</tr>
<tr>
<td>annotation_map</td>
<td>17</td>
</tr>
<tr>
<td>annotation_raster</td>
<td>18</td>
</tr>
<tr>
<td>autoplot</td>
<td>19</td>
</tr>
<tr>
<td>borders</td>
<td>19</td>
</tr>
<tr>
<td>coord_cartesian</td>
<td>20</td>
</tr>
<tr>
<td>coord_fixed</td>
<td>21</td>
</tr>
<tr>
<td>coord_flip</td>
<td>22</td>
</tr>
<tr>
<td>coord_map</td>
<td>23</td>
</tr>
<tr>
<td>coord_polar</td>
<td>25</td>
</tr>
<tr>
<td>coord_trans</td>
<td>26</td>
</tr>
<tr>
<td>cut_interval</td>
<td>28</td>
</tr>
<tr>
<td>diamonds</td>
<td>29</td>
</tr>
<tr>
<td>economics</td>
<td>30</td>
</tr>
<tr>
<td>expand_limits</td>
<td>31</td>
</tr>
<tr>
<td>facet_grid</td>
<td>31</td>
</tr>
<tr>
<td>facet_wrap</td>
<td>34</td>
</tr>
<tr>
<td>faithful</td>
<td>36</td>
</tr>
<tr>
<td>fortify</td>
<td>36</td>
</tr>
<tr>
<td>geom_abline</td>
<td>37</td>
</tr>
<tr>
<td>geom_bar</td>
<td>39</td>
</tr>
<tr>
<td>geom_bin2d</td>
<td>42</td>
</tr>
<tr>
<td>geom_blank</td>
<td>43</td>
</tr>
<tr>
<td>geom_boxplot</td>
<td>44</td>
</tr>
<tr>
<td>geom_contour</td>
<td>48</td>
</tr>
<tr>
<td>geom_count</td>
<td>50</td>
</tr>
<tr>
<td>geom_crossbar</td>
<td>52</td>
</tr>
<tr>
<td>geom_density</td>
<td>55</td>
</tr>
<tr>
<td>geom_density_2d</td>
<td>57</td>
</tr>
<tr>
<td>geom_dotplot</td>
<td>59</td>
</tr>
<tr>
<td>geom_errorbarh</td>
<td>62</td>
</tr>
<tr>
<td>geom_freqpoly</td>
<td>64</td>
</tr>
<tr>
<td>geom_hex</td>
<td>67</td>
</tr>
<tr>
<td>geom_jitter</td>
<td>69</td>
</tr>
<tr>
<td>geom_label</td>
<td>71</td>
</tr>
<tr>
<td>geom_map</td>
<td>75</td>
</tr>
<tr>
<td>geom_path</td>
<td>77</td>
</tr>
<tr>
<td>R topic documentated</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>geom_point</td>
<td>80</td>
</tr>
<tr>
<td>geom_polygon</td>
<td>83</td>
</tr>
<tr>
<td>geom_qq</td>
<td>85</td>
</tr>
<tr>
<td>geom_quantile</td>
<td>87</td>
</tr>
<tr>
<td>geom_raster</td>
<td>89</td>
</tr>
<tr>
<td>geom_ribbon</td>
<td>91</td>
</tr>
<tr>
<td>geom_rug</td>
<td>93</td>
</tr>
<tr>
<td>geom_segment</td>
<td>95</td>
</tr>
<tr>
<td>geom_smooth</td>
<td>97</td>
</tr>
<tr>
<td>geom_spoke</td>
<td>100</td>
</tr>
<tr>
<td>geom_violin</td>
<td>102</td>
</tr>
<tr>
<td>ggplot</td>
<td>105</td>
</tr>
<tr>
<td>ggproto</td>
<td>107</td>
</tr>
<tr>
<td>ggsave</td>
<td>108</td>
</tr>
<tr>
<td>ggttheme</td>
<td>109</td>
</tr>
<tr>
<td>guides</td>
<td>111</td>
</tr>
<tr>
<td>guide_colourbar</td>
<td>112</td>
</tr>
<tr>
<td>guide_legend</td>
<td>115</td>
</tr>
<tr>
<td>hmisc</td>
<td>118</td>
</tr>
<tr>
<td>labeller</td>
<td>119</td>
</tr>
<tr>
<td>labellers</td>
<td>121</td>
</tr>
<tr>
<td>label_bquote</td>
<td>123</td>
</tr>
<tr>
<td>labs</td>
<td>124</td>
</tr>
<tr>
<td>lims</td>
<td>125</td>
</tr>
<tr>
<td>luv_colours</td>
<td>126</td>
</tr>
<tr>
<td>margin</td>
<td>126</td>
</tr>
<tr>
<td>mean_se</td>
<td>128</td>
</tr>
<tr>
<td>midwest</td>
<td>129</td>
</tr>
<tr>
<td>mpg</td>
<td>130</td>
</tr>
<tr>
<td>msleep</td>
<td>131</td>
</tr>
<tr>
<td>position_dodge</td>
<td>132</td>
</tr>
<tr>
<td>position_identity</td>
<td>133</td>
</tr>
<tr>
<td>position_jitter</td>
<td>133</td>
</tr>
<tr>
<td>position_jitterdodge</td>
<td>134</td>
</tr>
<tr>
<td>position_nudge</td>
<td>135</td>
</tr>
<tr>
<td>position_stack</td>
<td>136</td>
</tr>
<tr>
<td>presidential</td>
<td>138</td>
</tr>
<tr>
<td>print.ggplot</td>
<td>139</td>
</tr>
<tr>
<td>print.ggproto</td>
<td>140</td>
</tr>
<tr>
<td>qplot</td>
<td>141</td>
</tr>
<tr>
<td>resolution</td>
<td>142</td>
</tr>
<tr>
<td>scale_alpha</td>
<td>143</td>
</tr>
<tr>
<td>scale_colour_brewer</td>
<td>144</td>
</tr>
<tr>
<td>scale_colour_gradient</td>
<td>146</td>
</tr>
<tr>
<td>scale_colour_hue</td>
<td>148</td>
</tr>
<tr>
<td>scale_continuous</td>
<td>149</td>
</tr>
<tr>
<td>scale_date</td>
<td>150</td>
</tr>
</tbody>
</table>
Description

+ is the key to constructing sophisticated ggplot2 graphics. It allows you to start simple, then get more and more complex, checking your work at each step.

Usage

```r
## S3 method for class 'gg'
e1 + e2
```

Arguments

- `e1` An object of class `ggplot` or a `theme`.
- `e2` A plot component, as described below.

What can you add?

You can add any of the following types of objects:

- A `aes()` object replaces the default aesthetics.
- A layer created by a `geom_` or `stat_` function adds a new layer.
- A scale overrides the existing scale.
• A `theme` modifies the current theme.
• A `coord` overrides current coordinate system.
• A `facet` specificatio override current faceting.

To replace the current default data frame, you must use `%>%`, due to S3 method precedence issues. You can also supply a list, in which case each element of the list will be added in turn.

**See Also**

`theme`

**Examples**

```r
base <- ggplot(mpg, aes(displ, hwy)) + geom_point()
base + geom_smooth()

# To override the data, you must use `%>%`
base %>% subset(mpg, fl == "p")

# Alternatively, you can add multiple components with a list.
# This can be useful to return from a function.
base + list(subset(mpg, fl == "p"), geom_smooth())
```

---

**aes**  
*Construct aesthetic mappings*

**Description**

Aesthetic mappings describe how variables in the data are mapped to visual properties (aesthetics) of geoms. Aesthetic mappings can be set in `ggplot2` and in individual layers.

**Usage**

```r
aes(x, y, ...)
```

**Arguments**

`x, y, ...`  
List of name value pairs giving aesthetics to map to variables. The names for `x` and `y` aesthetics are typically omitted because they are so common; all other aesthetics must be named.

**Details**

This function also standardise aesthetic names by performing partial matching, converting color to `colour`, and translating old style R names to ggplot names (eg. `pch` to `shape`, `cex` to `size`)

**See Also**

See `aes_` for a version of `aes` that is more suitable for programming with.
Examples

eaes(x = mpg, y = wt)
eaes(mpg, wt)

# You can also map aesthetics to functions of variables
aes(x = mpg * 2, y = wt / cyl)

# Aesthetic names are automatically standardised
aes(col = x)
eaes(fg = x)
eaes(color = x)
eaes(colour = x)

# aes is almost always used with ggplot() or a layer
ggplot(mpg, aes(displ, hwy)) + geom_point()

# Aesthetics supplied to ggplot() are used as defaults for every layer
# you can override them, or supply different aesthetics for each layer

---

Definition

Aesthetic mappings describe how variables in the data are mapped to visual properties (aesthetics) of geoms. `aes` uses non-standard evaluation to capture the variable names. `aes_` and `aes_string` require you to explicitly quote the inputs either with ` BB quote BB` for `aes_string()`, or with `quote` or `~` for `aes()`. (`aes_q` is an alias to `aes_`). This makes `aes_` and `aes_string` easy to program with.

Usage

`aes_(x, y, ...)`

`aes_string(x, y, ...)`

`aes_q(x, y, ...)`

Arguments

- `x, y, ...`: List of name value pairs. Elements must be either quoted calls, strings, one-sided formulas or constants.

Details

`aes_string` and `aes_` are particularly useful when writing functions that create plots because you can use strings or quoted names/calls to define the aesthetic mappings, rather than having to use `substitute` to generate a call to `aes()`.
I recommend using `aes_()` because creating the equivalents of `aes(colour = "my colour")` or `aes(x = `X$1`)` with `aes_string()` is quite clunky.

**See Also**

`aes`

**Examples**

```r
# Three ways of generating the same aesthetics
aes(mpg, wt, col = cyl)
aes_(quote(mpg), quote(wt), col = quote(cyl))
aes_(~mpg, ~wt, col = ~cyl)
aes_string("mpg", "wt", col = "cyl")

# You can’t easily mimic these calls with `aes_string`
aes("$100", colour = "smooth")
aes_(~ "$100", colour = "smooth")
# Ok, you can, but it requires a _lot_ of quotes
aes_string(""$100"", colour = "smooth")

# Convert strings to names with `as_name`
var <- "cyl"
aes(col = x)
aes_(col = as.name(var))
```

---

**Description**

This page demonstrates the usage of a sub-group of aesthetics: colour, fill and alpha.

**Examples**

```r
# Bar chart example
c <- ggplot(mtcars, aes(factor(cyl)))
# Default plotting
c + geom_bar()
# To change the interior colouring use `fill` aesthetic
nc + geom_bar(fill = "red")
# Compare with the colour aesthetic which changes just the bar outline
nc + geom_bar(colour = "red")
# Combining both, you can see the changes more clearly
nc + geom_bar(fill = "white", colour = "red")

# The aesthetic `fill` also takes different colouring scales
# setting `fill` equal to a factor variable uses a discrete colour scale
```
## aes_group_order

```r
k <- ggplot(mtcars, aes(factor(cyl), fill = factor(vs)))
k + geom_bar()

# Fill aesthetic can also be used with a continuous variable
m <- ggplot(faithful, aes(waiting, eruptions))
m + geom_raster()
m + geom_raster(aes(fill = density))

# Some geoms don't use both aesthetics (i.e. geom_point or geom_line)
b <- ggplot(economics, aes(x = date, y = unemploy))
b + geom_line()
b + geom_line(colour = "green")
b + geom_point()
b + geom_point(colour = "red")

# For large datasets with overplotting the alpha
# aesthetic will make the points more transparent
df <- data.frame(x = rnorm(5000), y = rnorm(5000))
h <- ggplot(df, aes(x, y))
h + geom_point()
h + geom_point(alpha = 0.5)
h + geom_point(alpha = 1/10)

# Alpha can also be used to add shading
j <- b + geom_line()

j yrng <- range(economics$unemploy)
j <- j + geom_rect(aes(NULL, NULL, xmin = start, xmax = end, fill = party),
ymin = yrng[1], ymax = yrng[2], data = presidential)
j
j + scale_fill_manual(values = alpha(c("blue", "red"), .3))
```

<table>
<thead>
<tr>
<th>aes_group_order</th>
<th>Aesthetics: grouping</th>
</tr>
</thead>
</table>

## Description

Aesthetics: grouping

## Examples

# By default, the group is set to the interaction of all discrete variables in the # plot. This often partitions the data correctly, but when it does not, or when # no discrete variable is used in the plot, you will need to explicitly define the # grouping structure, by mapping group to a variable that has a different value # for each group.
For most applications you can simply specify the grouping with various aesthetics (colour, shape, fill, linetype) or with facets.

```r
p <- ggplot(mtcars, aes(wt, mpg))
# A basic scatter plot
p + geom_point(size = 4)
# The colour aesthetic
p + geom_point(aes(colour = factor(cyl)), size = 4)
# Or you can use shape to distinguish the data
p + geom_point(aes(shape = factor(cyl)), size = 4)

# Using fill
a <- ggplot(mtcars, aes(factor(cyl)))
a + geom_bar()
a + geom_bar(aes(fill = factor(cyl)))
a + geom_bar(aes(fill = factor(vs)))

# Using linetypes
rescale <- function(x) (x - min(x)) / diff(range(x))
ec_scaled <- data.frame(
  date = economics$date,
  pplyr::colwise(rescale)(economics[, -c(1:2)]))
ecm <- reshape2::melt(ec_scaled, id.vars = "date")
f <- ggplot(ecm, aes(date, value))
f + geom_line(aes(linetype = variable))

# Using facets
k <- ggplot(diamonds, aes(carat, ..density..)) + geom_histogram(binwidth = 0.2)
k + facet_grid(~ cut)
```

There are three common cases where the default is not enough, and we will consider each one below. In the following examples, we will use a simple longitudinal dataset, Oxboys, from the nlme package. It records the heights (height) and centered ages (age) of 26 boys (Subject), measured on nine occasions (Occasion).

# Multiple groups with one aesthetic
h <- ggplot(nlme::Oxboys, aes(age, height))
# A single line tries to connect all the observations
h + geom_line()
# The group aesthetic maps a different line for each subject
h + geom_line(aes(group = Subject))

# Different groups on different layers
h <- h + geom_line(aes(group = Subject))
# Using the group aesthetic with both geom_line() and geom_smooth()
# groups the data the same way for both layers
h + geom_smooth(aes(group = Subject), method = "lm", se = FALSE)
# Changing the group aesthetic for the smoother layer
# fits a single line of best fit across all boys
h + geom_smooth(aes(group = 1), size = 2, method = "lm", se = FALSE)

# Overriding the default grouping
# The plot has a discrete scale but you want to draw lines that connect across # groups. This is the strategy used in interaction plots, profile plots, and parallel # coordinate plots, among others. For example, we draw boxplots of height at # each measurement occasion
boysbox <- ggplot(nlme:::Oxboys, aes(occasion, height))
boysbox + geom_boxplot()
# There is no need to specify the group aesthetic here; the default grouping # works because occasion is a discrete variable. To overlay individual trajectories # we again need to override the default grouping for that layer with aes(group = Subject)
boysbox <- boysbox + geom_boxplot()
boysbox + geom_line(aes(group = Subject), colour = "blue")

### Description

This page demonstrates the usage of a sub-group of aesthetics; linetype, size and shape.

### Examples

# Line types should be specified with either an integer, a name, or with a string of # an even number (up to eight) of hexadecimal digits which give the lengths in # consecutive positions in the string.
# 0 = blank, 1 = solid, 2 = dashed, 3 = dotted, 4 = dotdash, 5 = longdash, 6 = twodash

# Data
df <- data.frame(x = 1:10, y = 1:10)
f <- ggplot(df, aes(x, y))
f + geom_line(linetype = 2)
f + geom_line(linetype = "dotdash")

# An example with hex strings, the string "33" specifies three units on followed # by three off and "3313" specifies three units on followed by three off followed # by one on and finally three off.
f + geom_line(linetype = "3313")

# Mapping line type from a variable
ggplot(economics_long, aes(date, value01)) +
geom_line(aes(linetype = variable))

# Size examples
# Should be specified with a numerical value (in millimetres), # or from a variable source
p <- ggplot(mtcars, aes(wt, mpg))
p + geom_point(size = 4)
p + geom_point(aes(size = qsec))
p + geom_point(size = 2.5) +
  geom_hline(yintercept = 25, size = 3.5)

# Shape examples
# Shape takes four types of values: an integer in [0, 25],
# a single character---which uses that character as the plotting symbol,
# a . to draw the smallest rectangle that is visible (i.e., about one pixel)
# an NA to draw nothing
p + geom_point()
p + geom_point(shape = 5)
p + geom_point(shape = "k", size = 3)
p + geom_point(shape = ".")
p + geom_point(shape = NA)

# Shape can also be mapped from a variable
p + geom_point(aes(shape = factor(cyl)))

# A look at all 25 symbols
df2 <- data.frame(x = 1:5, y = 1:25, z = 1:25)
s <- ggplot(df2, aes(x, y))
s + geom_point(aes(shape = z), size = 4) +
  scale_shape_identity()
# While all symbols have a foreground colour, symbols 19-25 also take a
# background colour (fill)
s + geom_point(aes(shape = z), size = 4, colour = "Red") +
  scale_shape_identity()
s + geom_point(aes(shape = z), size = 4, colour = "Red", fill = "Black") +
  scale_shape_identity()

### aes_position

**Position related aesthetics: x, y, xmin, xmax, ymin, ymax, xend, yend**

**Description**

This page demonstrates the usage of a sub-group of aesthetics; x, y, xmin, xmax, ymin, ymax, xend, and yend.

**Examples**

# Generate data: means and standard errors of means for prices
# for each type of cut
dmod <- lm(price ~ cut, data = diamonds)
cuts <- data.frame(cut = unique(diamonds$cut), predict(dmod, data.frame(cut =
unique(diamonds$cut)), se = TRUE)[c("fit", "se.fit")])
se <- ggplot(cuts, aes(x = cut, y = fit, ymin = fit - se.fit,
  ymax = fit + se.fit, colour = cut))
se + geom_pointrange()

# Using annotate
annotate

Create an annotation layer

Description

This function adds geoms to a plot, but unlike typical a geom function, the properties of the geoms are not mapped from variables of a data frame, but are instead passed in as vectors. This is useful for adding small annotations (such as text labels) or if you have your data in vectors, and for some reason don’t want to put them in a data frame.

Usage

annotate(geom, x = NULL, y = NULL, xmin = NULL, xmax = NULL, ymin = NULL, ymax = NULL, xend = NULL, yend = NULL, ..., na.rm = FALSE)

Arguments

geom name of geom to use for annotation
x, y, xmin, ymin, xmax, ymax, xend, yend positioning aesthetics - you must specify at least one of these.
... other arguments passed on to layer. These are often aesthetics, used to set an aesthetic to a fixed value, like color = "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.
Details

Note that all position aesthetics are scaled (i.e. they will expand the limits of the plot so they are visible), but all other aesthetics are set. This means that layers created with this function will never affect the legend.

Examples

```r
p <- ggplot(mtcars, aes(x = wt, y = mpg)) + geom_point()
p + annotate("text", x = 4, y = 25, label = "Some text")
p + annotate("rect", x = 2:5, y = 25, label = "Some text")
p + annotate("rect", xmin = 3, xmax = 4.2, ymin = 12, ymax = 21, alpha = .2)
p + annotate("segment", x = 2.5, xend = 4, y = 15, yend = 25, colour = "blue")
p + annotate("pointrange", x = 3.5, y = 20, xmin = 12, xmax = 28, colour = "red", size = 1.5)
p + annotate("text", x = 2:3, y = 20:21, label = c("my label", "label 2"))
p + annotate("text", x = 4, y = 25, label = "italic(R) ^ 2 == 0.75", parse = TRUE)
p + annotate("text", x = 4, y = 25, label = "paste(italic(R) ^ 2, \"r\" = .75\")", parse = TRUE)
```

Description

This is a special geom intended for use as static annotations that are the same in every panel. These annotations will not affect scales (i.e. the x and y axes will not grow to cover the range of the grob, and the grob will not be modified by any ggplot settings or mappings).

Usage

```r
annotation_custom(grob, xmin = -Inf, xmax = Inf, ymin = -Inf, ymax = Inf)
```

Arguments

- `grob`: grob to display
- `xmin, xmax`: x location (in data coordinates) giving horizontal location of raster
- `ymin, ymax`: y location (in data coordinates) giving vertical location of raster

Details

Most useful for adding tables, inset plots, and other grid-based decorations.
Note

annotation_custom expects the grob to fill the entire viewport defined by xmin, xmax, ymin, ymax. Grobs with a different (absolute) size will be center-justified in that region. Inf values can be used to fill the full plot panel (see examples).

Examples

# Dummy plot
df <- data.frame(x = 1:10, y = 1:10)
base <- ggplot(df, aes(x, y)) +
  geom_blank() +
  theme_bw()

# Full panel annotation
base + annotation_custom(
  grob = grid::roundrectGrob(),
  xmin = -Inf, xmax = Inf, ymin = -Inf, ymax = Inf
)

# Inset plot
df2 <- data.frame(x = 1, y = 1)
g <- ggplotGrob(ggplot(df2, aes(x, y)) +
  geom_point() +
  theme(plot.background = element_rect(colour = "black")))
base +
  annotation_custom(grob = g, xmin = 1, xmax = 10, ymin = 8, ymax = 10)

---

annotation_logticks   Annotation: log tick marks

Description

This annotation adds log tick marks with diminishing spacing. These tick marks probably make sense only for base 10.

Usage

annotation_logticks(base = 10, sides = "bl", scaled = TRUE,
short = unit(0.1, "cm"), mid = unit(0.2, "cm"), long = unit(0.3, "cm"),
colour = "black", size = 0.5, linetype = 1, alpha = 1, color = NULL,
...)

Arguments

base     the base of the log (default 10)
sides    a string that controls which sides of the plot the log ticks appear on. It can be set to a string containing any of "trbl", for top, right, bottom, and left.
is the data already log-scaled? This should be TRUE (default) when the data is already transformed with log10() or when using scale_y_log10. It should be FALSE when using coord_trans(y = "log10").

short

a unit object specifying the length of the short tick marks

mid

a unit object specifying the length of the middle tick marks. In base 10, these are the "5" ticks.

long

a unit object specifying the length of the long tick marks. In base 10, these are the "1" (or "10") ticks.

colour

Colour of the tick marks.

size

Thickness of tick marks, in mm.

linetype

Linetype of tick marks (solid, dashed, etc.)

alpha

The transparency of the tick marks.

color

An alias for colour.

... Other parameters passed on to the layer

See Also

scale_y_continuous, scale_y_log10 for log scale transformations.

coord_trans for log coordinate transformations.

Examples

# Make a log-log plot (without log ticks)
a <- ggplot(msleep, aes(bodywt, brainwt)) +
  geom_point(na.rm = TRUE) +
  scale_x_log10(
    breaks = scales::trans_breaks("log10", function(x) 10^x),
    labels = scales::trans_format("log10", scales::math_format(10^x))
  ) +
  scale_y_log10(
    breaks = scales::trans_breaks("log10", function(x) 10^x),
    labels = scales::trans_format("log10", scales::math_format(10^x))
  ) +
  theme_bw()

a + annotation_logticks()  # Default: log ticks on bottom and left
a + annotation_logticks(sides = "lr")  # Log ticks for y, on left and right
a + annotation_logticks(sides = "trbl")  # All four sides

# Hide the minor grid lines because they don’t align with the ticks
a + annotation_logticks(sides = "trbl") + theme(panel.grid.minor = element_blank())

# Another way to get the same results as ’a’ above: log-transform the data before
# plotting it. Also hide the minor grid lines.
b <- ggplot(msleep, aes(log10(bodywt), log10(brainwt))) +
  geom_point(na.rm = TRUE) +
  scale_x_continuous(name = "body", labels = scales::math_format(10^x)) +
  scale_y_continuous(name = "brain", labels = scales::math_format(10^x)) +
theme_bw() + theme(panel.grid.minor = element_blank())

b + annotation_logticks()

# Using a coordinate transform requires scaled = FALSE
t <- ggplot(msleep, aes(bodywt, brainwt)) +
  geom_point() +
  coord_trans(x = "log10", y = "log10") +
  theme_bw()
t + annotation_logticks(scaled = FALSE)

# Change the length of the ticks
a + annotation_logticks(
  short = unit(.5,"mm"),
  mid = unit(3,"mm"),
  long = unit(4,"mm"))

---

**annotation_map**

Annotation: a maps

**Description**

Display a fixed map on a plot.

**Usage**

annotation_map(map, ...)

**Arguments**

- **map**: data frame representing a map. Most map objects can be converted into the right format by using `fortify`
- **...**: other arguments used to modify aesthetics

**Examples**

```r
if (require("maps")) {
  usamap <- map_data("state")

  seal.sub <- subset(seals, long > -130 & lat < 45 & lat > 40)
  ggplot(seal.sub, aes(x = long, y = lat)) +
    annotation_map(usamap, fill = "NA", colour = "grey50") +
    geom_segment(aes(xend = long + delta_long, yend = lat + delta_lat))

  seal2 <- transform(seal.sub,
    latr = cut(lat, 2),
    longr = cut(long, 2))
```
ggplot2, `aes(x = long, y = lat)`) +
    annotation_map(usamap, fill = "NA", colour = "grey50") +
    geom_segment(aes(xend = long + delta_long, yend = lat + delta_lat)) +
    facet_grid(lat ~ long, scales = "free", space = "free")
}

---

**annotation_raster**  
**Annotation: high-performance rectangular tiling**

**Description**

This is a special version of `geom_raster` optimised for static annotations that are the same in every panel. These annotations will not affect scales (i.e. the x and y axes will not grow to cover the range of the raster, and the raster must already have its own colours). This is useful for adding bitmap images.

**Usage**

```r
annotation_raster(raster, xmin, xmax, ymin, ymax, interpolate = FALSE)
```

**Arguments**

- `raster`: raster object to display
- `xmin, xmax`: x location (in data coordinates) giving horizontal location of raster
- `ymin, ymax`: y location (in data coordinates) giving vertical location of raster
- `interpolate`: If TRUE interpolate linearly, if FALSE (the default) don’t interpolate.

**Examples**

```r
# Generate data
rainbow <- matrix(hcl(seq(0, 360, length.out = 50 * 50), 80, 70), nrow = 50)
ggplot(mtcars, aes(mpg, wt)) +
  geom_point() +
  annotation_raster(rainbow, 15, 20, 3, 4)

# To fill up whole plot
rainbow2 <- matrix(hcl(seq(0, 360, length.out = 10), 80, 70), nrow = 1)
ggplot(mtcars, aes(mpg, wt)) +
  annotation_raster(rainbow2, -Inf, Inf, -Inf, Inf) +
  geom_point()
```

```r
rainbow2 <- matrix(hcl(seq(0, 360, length.out = 10), 80, 70), nrow = 1)
ggplot(mtcars, aes(mpg, wt)) +
  annotation_raster(rainbow2, -Inf, Inf, -Inf, Inf, interpolate = TRUE) +
  geom_point()
```
**autoplot**

Create a complete ggplot appropriate to a particular data type

**Description**

autoplot uses ggplot2 to draw a particular plot for an object of a particular class in a single command. This defines the S3 generic that other classes and packages can extend.

**Usage**

autoplot(object, ...)

**Arguments**

- `object` an object, whose class will determine the behaviour of autoplot
- `...` other arguments passed to specific methods

**Value**

a ggplot object

**See Also**

[ggplot](https://ggplot2.tidyverse.org) and [fortify](https://www.rfordatascience.org/tidyverse/)

---

**borders**

Create a layer of map borders

**Description**

This is a quick and dirty way to get map data (from the maps package) on to your plot. This is a good place to start if you need some crude reference lines, but you'll typically want something more sophisticated for communication graphics.

**Usage**

borders(database = "world", regions = ".", fill = NA, colour = "grey50", xlim = NULL, ylim = NULL, ...)

**Arguments**

- `database` map data, see map for details
- `regions` map region
- `fill` fill colour
- `colour` border colour
- `xlim, ylim` latitudinal and longitudinal range for extracting map polygons, see map for details.
- `...` other arguments passed onto geom_polygon
Examples

```r
if (require("maps")) {

  ia <- map_data("county", "iowa")
  mid_range <- function(x) mean(range(x))
  seats <- plyr::ddply(ia, "subregion", plyr::colwise(mid_range, c("lat", "long")))
  ggplot(ia, aes(long, lat)) +
    geom_polygon(aes(group = group), fill = NA, colour = "grey60") +
    geom_text(aes(label = subregion), data = seats, size = 2, angle = 45)

  data(us.cities)
  capitals <- subset(us.cities, capital == 2)
  ggplot(capitals, aes(long, lat)) +
    borders("state") +
    geom_point(aes(size = pop)) +
    scale_size_area() +
    coord_quickmap()

  # Same map, with some world context
  ggplot(capitals, aes(long, lat)) +
    borders("world", xlim = c(-130, -60), ylim = c(20, 50)) +
    geom_point(aes(size = pop)) +
    scale_size_area() +
    coord_quickmap()
}
```

---

**coord_cartesian**

*Cartesian coordinates*

**Description**

The Cartesian coordinate system is the most familiar, and common, type of coordinate system. 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.

**Usage**

```r
coord_cartesian(xlim = NULL, ylim = NULL, expand = TRUE)
```

**Arguments**

- `xlim, ylim` Limits for the x and y axes.
- `expand` If TRUE, the default, adds a small expansion factor to the limits to ensure that data and axes don’t overlap. If FALSE, limits are taken exactly from the data or xlim/ylim.
Examples

```r
p <- ggplot(mtcars, aes(disp, wt)) +
  geom_point() +
  geom_smooth()
p

# Setting the limits on a scale converts all values outside the range to NA.
p + scale_x_continuous(limits = c(325, 500))

# Setting the limits on the coordinate system performs a visual zoom.
# The data is unchanged, and we just view a small portion of the original
# plot. Note how smooth continues past the points visible on this plot.
p + coord_cartesian(xlim = c(325, 500))

# By default, the same expansion factor is applied as when setting scale
# limits. You can set the limits precisely by setting expand = FALSE
p + coord_cartesian(xlim = c(325, 500), expand = FALSE)

# Similarly, we can use expand = FALSE to turn off expansion with the
# default limits
p + coord_cartesian(expand = FALSE)

# You can see the same thing with this 2d histogram
d <- ggplot(diamonds, aes(carat, price)) +
  stat_bin2d(bins = 25, colour = "white")
d
# When zooming the scale, the we get 25 new bins that are the same
# size on the plot, but represent smaller regions of the data space
d + scale_x_continuous(limits = c(0, 1))

# When zooming the coordinate system, we see a subset of original 50 bins,
# displayed bigger
d + coord_cartesian(xlim = c(0, 1))
```

---

**coord_fixed**

**Cartesian coordinates with fixed "aspect ratio"**

Description

A fixed scale coordinate system 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. The default, ratio = 1, ensures that one unit on the x-axis is the same length as one unit on the y-axis. Ratios higher than one make units on the y axis longer than units on the x-axis, and vice versa. This is similar to eqscplot, but it works for all types of graphics.
Usage

coord_fixed(ratio = 1, xlim = NULL, ylim = NULL, expand = TRUE)

Arguments

ratio aspect ratio, expressed as $y / x$
xlim Limits for the x and y axes.
ylim Limits for the x and y axes.
expand If TRUE, the default, adds a small expansion factor to the limits to ensure that data and axes don’t overlap. If FALSE, limits are taken exactly from the data or xlim/ylim.

Examples

# ensures that the ranges of axes are equal to the specified ratio by
# adjusting the plot aspect ratio

p <- ggplot(mtcars, aes(mpg, wt)) + geom_point()
p + coord_fixed(ratio = 1)
p + coord_fixed(ratio = 5)
p + coord_fixed(ratio = 1/5)

# Resize the plot to see that the specified aspect ratio is maintained

---

coord_flip

*Cartesian coordinates with x and y flipped*

Description

Flip cartesian coordinates so that horizontal becomes vertical, and vertical, horizontal. This is primarily useful for converting geoms and statistics which display y conditional on x, to x conditional on y.

Usage

coord_flip(xlim = NULL, ylim = NULL, expand = TRUE)

Arguments

xlim Limits for the x and y axes.
ylim Limits for the x and y axes.
expand If TRUE, the default, adds a small expansion factor to the limits to ensure that data and axes don’t overlap. If FALSE, limits are taken exactly from the data or xlim/ylim.
Examples

# Very useful for creating boxplots, and other interval
geoms in the horizontal instead of vertical position.

ggplot(diamonds, aes(cut, price)) +
geom_boxplot() +
coord_flip()

h <- ggplot(diamonds, aes(carat)) +
geom_histogram()
h + coord_flip()
h + coord_flip() + scale_x_reverse()

# You can also use it to flip line and area plots:
df <- data.frame(x = 1:5, y = (1:5) ^ 2)
ggplot(df, aes(x, y)) +
geom_area()last_plot() + coord_flip()
xlim, ylim  Manually specific x/y limits (in degrees of longitude/latitude)
expand  If TRUE, the default, adds a small expansion factor to the limits to ensure that data and axes don’t overlap. If FALSE, limits are taken exactly from the data or xlim/ylim.

Details

In general, map projections must account for the fact that the actual length (in km) of one degree of longitude varies between the equator and the pole. Near the equator, the ratio between the lengths of one degree of latitude and one degree of longitude is approximately 1. Near the pole, it is tends towards infinity because the length of one degree of longitude tends towards 0. For regions that span only a few degrees and are not too close to the poles, setting the aspect ratio of the plot to the appropriate lat/lon ratio approximates the usual mercator projection. This is what coord_quickmap does, and is much faster (particularly for complex plots like geom_tile) at the expense of correctness.

Examples

```r
if (require("maps")) {
  nz <- map_data("nz")
  # Prepare a map of NZ
  nzmap <- ggplot(nz, aes(x = long, y = lat, group = group)) +
           geom_polygon(fill = "white", colour = "black")
  # Plot it in cartesian coordinates
  nzmap
  # With correct mercator projection
  nzmap + coord_map()
  # With the aspect ratio approximation
  nzmap + coord_quickmap()

  # Other projections
  nzmap + coord_quickmap("cylindrical")
  nzmap + coord_quickmap("azequalarea", orientation = c(-36.92, 174.6, 0))
  nzmap + coord_quickmap("lambert", parameters = c(-37, -44))

  states <- map_data("state")
  usamap <- ggplot(states, aes(long, lat, group = group)) +
            geom_polygon(fill = "white", colour = "black")
  # Use cartesian coordinates
  usamap
  # With mercator projection
  usamap + coord_map()
  usamap + coord_quickmap()
  # See ?mapproject for coordinate systems and their parameters
  usamap + coord_quickmap("gilbert")
  usamap + coord_quickmap("lagrange")

  # For most projections, you'll need to set the orientation yourself
  # as the automatic selection done by mapproject is not available to
```
# ggplot
usamap + coord_map("orthographic")
usamap + coord_map("stereographic")
usamap + coord_map("conic", lat0 = 30)
usamap + coord_map("bonne", lat0 = 50)

# World map, using geom_path instead of geom_polygon
world <- map_data("world")
worldmap <- ggplot(world, aes(x = long, y = lat, group = group)) +
  geom_path() +
  scale_y_continuous(breaks = (-2:2) * 30) +
  scale_x_continuous(breaks = (-4:4) * 45)

# Orthographic projection with default orientation (looking down at North pole)
worldmap + coord_map("ortho")
# Looking up up at South Pole
worldmap + coord_map("ortho", orientation = c(-90, 0, 0))
# Centered on New York (currently has issues with closing polygons)
worldmap + coord_map("ortho", orientation = c(41, -74, 0))

---

**coord_polar**

*Polar coordinates*

**Description**

The polar coordinate system is most commonly used for pie charts, which are a stacked bar chart in polar coordinates.

**Usage**

coord_polar(theta = "x", start = 0, direction = 1)

**Arguments**

- `theta` variable to map angle to (`x` or `y`)
- `start` offset of starting point from 12 o’clock in radians
- `direction` 1, clockwise; -1, anticlockwise

**Examples**

# NOTE: Use these plots with caution - polar coordinates has
# major perceptual problems. The main point of these examples is
# to demonstrate how these common plots can be described in the
# grammar. Use with EXTREME caution.

#’ # A pie chart = stacked bar chart + polar coordinates
pie <- ggplot(mtcars, aes(x = factor(1), fill = factor(cyl))) +
  geom_bar(width = 1)
pie + coord_polar(theta = "y")

# A coxcomb plot = bar chart + polar coordinates
cxc <- ggplot(mtcars, aes(x = factor(cyl))) +
    geom_bar(width = 1, colour = "black")
cxc + coord_polar()
# A new type of plot?
cxc + coord_polar(theta = "y")

# The bullseye chart
pie + coord_polar()

# Hadley's favourite pie chart
df <- data.frame(
    variable = c("does not resemble", "resembles"),
    value = c(20, 80)
)
ggplot(df, aes(x = ",", y = value, fill = variable)) +
    geom_col(width = 1) +
    scale_fill_manual(values = c("red", "yellow")) +
    coord_polar("y", start = pi / 3) +
    labs(title = "Pacman")

# Windrose + doughnut plot
if (require("ggplot2movies")) {
    movies$rating <- cut_interval(movies$rating, length = 1)
    movies$budgetq <- cut_number(movies$budget, 4)

doh <- ggplot(movies, aes(x = rrating, fill = budgetq))

# Wind rose
doh + geom_bar(width = 1) + coord_polar()
# Race track plot
doh + geom_bar(width = 0.9, position = "fill") + coord_polar(theta = "y")
}

---

**coord_trans**  
*Transformed Cartesian coordinate system*

**Description**

`coord_trans` is different to 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.
Usage
coord_trans(x = "identity", y = "identity", limx = NULL, limy = NULL,
xtrans, ytrans)

Arguments
x, y transformers for x and y axes
limx, limy limits for x and y axes. (Named so for backward compatibility)
xtrans, ytrans Deprecated; use x and y instead.

Details
Transformations only work with continuous values: see trans_new for list of transformations, and
instructions on how to create your own.

Examples

# See ?geom_boxplot for other examples

# Three ways of doing transformation in ggplot:
# * by transforming the data
ggplot(diamonds, aes(log10(carat), log10(price))) +
  geom_point()
# * by transforming the scales
ggplot(diamonds, aes(carat, price)) +
  geom_point() +
  scale_x_log10() +
  scale_y_log10()
# * by transforming the coordinate system:
  coord_trans(x = "log10", y = "log10")

# The difference between transforming the scales and
# transforming the coordinate system is that scale
# transformation occurs BEFORE statistics, and coordinate
# transformation afterwards. Coordinate transformation also
# changes the shape of geoms:
d <- subset(diamonds, carat > 0.5)

  ggplot(d, aes(carat, price)) +
    geom_point() +
    geom_smooth(method = "lm") +
    scale_x_log10() +
    scale_y_log10()
cut_interval

Discretise numeric data into categorical

Description

cut_interval makes \( n \) groups with equal range, cut_number makes \( n \) groups with (approximately) equal numbers of observations; cut_width makes groups of width \( width \).

Usage

```r
\texttt{cut_interval}(x, n = \texttt{NULL}, length = \texttt{NULL}, \ldots)
```

```r
\texttt{cut_number}(x, n = \texttt{NULL}, \ldots)
```

```r
\texttt{cut_width}(x, width, center = \texttt{NULL}, boundary = \texttt{NULL}, closed = \texttt{c("right", "left")})
```

Arguments

\( x \) numeric vector

\( n \) number of intervals to create, OR
length  length of each interval

...  other arguments passed on to `cut`

width  The bin width.

center, boundary

Specify either the position of edge or the center of a bin. Since all bins are
aligned, specifying the position of a single bin (which doesn’t need to be in the
range of the data) affects the location of all bins. If not specified, uses the "tile
layers algorithm", and sets the boundary to half of the binwidth.

To center on integers, width = 1 and center = 0, boundary = 0.5.

closed  One of "right" or "left" indicating whether right or left edges of bins are
included in the bin.

Author(s)

Randall Prium contributed most of the implementation of `cut_width`.

See Also

cut_number

Examples

table(cut_interval(1:100, 10))
table(cut_interval(1:100, 11))

table(cut_number(runif(1000), 10))

table(cut_width(runif(1000), 0.1))
table(cut_width(runif(1000), 0.1, boundary = 0))
table(cut_width(runif(1000), 0.1, center = 0))

diamonds  Prices of 50,000 round cut diamonds

Description

A dataset containing the prices and other attributes of almost 54,000 diamonds. The variables are
as follows:

Usage

diamonds
Format
A data frame with 53940 rows and 10 variables:

- price  price in US dollars ($326–$18,823)
- carat  weight of the diamond (0.2–5.01)
- cut  quality of the cut (Fair, Good, Very Good, Premium, Ideal)
- color  diamond colour, from J (worst) to D (best)
- clarity  a measurement of how clear the diamond is (I1 (worst), SI1, SI2, VS1, VS2, VVS1, VVS2, IF (best))
- x  length in mm (0–10.74)
- y  width in mm (0–58.9)
- z  depth in mm (0–31.8)
- depth  total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43–79)
- table  width of top of diamond relative to widest point (43–95)

Description
This dataset was produced from US economic time series data available from http://research.stlouisfed.org/fred2. economics is in "wide" format, economics_long is in "long" format.

Usage
economics

economics_long

Format
A data frame with 478 rows and 6 variables

date  Month of data collection
psavert  personal savings rate, http://research.stlouisfed.org/fred2/series/PSAVERT/
pce  personal consumption expenditures, in billions of dollars, http://research.stlouisfed.org/fred2/series/PCE
unemploy  number of unemployed in thousands, http://research.stlouisfed.org/fred2/series/UNEMPLOY
uempmed  median duration of unemployment, in weeks, http://research.stlouisfed.org/fred2/series/UEMPMED
pop  total population, in thousands, http://research.stlouisfed.org/fred2/series/POP
**expand_limits**

*Expand the plot limits, using data*

**Description**

Sometimes you may want to ensure limits include a single value, for all panels or all plots. This function is a thin wrapper around `geom_blank` that makes it easy to add such values.

**Usage**

```r
expand_limits(...)  
```

**Arguments**

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

**Examples**

```r
p <- ggplot(mtcars, aes(mpg, wt)) + geom_point()  
p + expand_limits(x = 0)  
p + expand_limits(y = c(1, 9))  
p + expand_limits(x = 0, y = 0)  

ggplot(mtcars, aes(mpg, wt)) +  
  geom_point(aes(colour = cyl)) +  
  expand_limits(colour = seq(2, 10, by = 2))  
ggplot(mtcars, aes(mpg, wt)) +  
  geom_point(aes(colour = factor(cyl))) +  
  expand_limits(colour = factor(seq(2, 10, by = 2)))
```

**facet_grid**

*Lay out panels in a grid*

**Description**

`facet_grid` forms a matrix of panels defined by row and column facetting variables. It is most useful when you have two discrete variables, and all combinations of the variables exist in the data.

**Usage**

```r
facet_grid(facets, margins = FALSE, scales = "fixed", space = "fixed",  
  shrink = TRUE, labeller = "label_value", as.table = TRUE,  
  switch = NULL, drop = TRUE)
```
Arguments

facets a formula with the rows (of the tabular display) on the LHS and the columns (of the tabular display) on the RHS; the dot in the formula is used to indicate there should be no faceting on this dimension (either row or column). The formula can also be provided as a string instead of a classical formula object.

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.

Examples

p <- ggplot(mpg, aes(displ, cty)) + geom_point()

p + facet_grid(~ cyl)
p + facet_grid(drv ~ .)
p + facet_grid(drv ~ cyl)

# To change plot order of facet grid,
# change the order of variable levels with factor()

# If you combine a facetted dataset with a dataset that lacks those
# facetting variables, the data will be repeated across the missing
facet_grid

```r
# combinations:
df <- data.frame(displ = mean(mpg$displ), cty = mean(mpg$cty))
p + facet_grid(. ~ cyl) +
  geom_point(data = df, colour = "red", size = 2)

# Free scales ---------------------------------------------
# You can also choose whether the scales should be constant
# across all panels (the default), or whether they should be allowed
# to vary
mt <- ggplot(mtcars, aes(mpg, wt, colour = factor(cyl))) +
  geom_point()

mt + facet_grid(. ~ cyl, scales = "free")

# If scales and space are free, then the mapping between position
# and values in the data will be the same across all panels. This
# is particularly useful for categorical axes
ggplot(mtcars, aes(driv, model)) +
  geom_point() +
  facet_grid(manufacturer ~ ., scales = "free", space = "free") +
  theme(strip.text.y = element_text(angle = 0))

# Facet labels ---------------------------------------------
p <- ggplot(mtcars, aes(wt, mpg)) + geom_point()
p

# label_both() displays both variable name and value
p + facet_grid(vs ~ cyl, labeller = label_both)

# label_parsed() parses text into mathematical expressions, see ?plotmath
mtcars$cyl2 <- factor(mtcars$cyl, labels = c("alpha", "beta", "sqrt(x, y)"))
ggplot(mtcars, aes(wt, mpg)) +
  geom_point() +
  facet_grid(. ~ cyl2, labeller = label_parsed)

# label_bquote() makes it easy to construct math expressions
p + facet_grid(. ~ vs, labeller = label_bquote(cols = alpha ^ .(vs)))

# The facet strips can be displayed near the axes with switch
data <- transform(mtcars,
  am = factor(am, levels = 0:1, c("Automatic", "Manual")),
  gear = factor(gear, levels = 3:5, labels = c("Three", "Four", "Five"))
)
p <- ggplot(data, aes(mpg, disp)) + geom_point()
p + facet_grid(am ~ gear, switch = "both")
# It looks better without boxes around the strips
p + facet_grid(am ~ gear, switch = "both") +
  theme(strip.background = element_blank())

# Margins ---------------------------------------------

# Margins can be specified by logically (all yes or all no) or by specific
# variables as (character) variable names
mg <- ggplot(mtcars, aes(x = mpg, y = wt)) + geom_point()
mg + facet_grid(vs + am ~ gear)
mg + facet_grid(vs + am ~ gear, margins = TRUE)
mg + facet_grid(vs + am ~ gear, margins = "am")

# when margins are made over "vs", since the facets for "am" vary
# within the values of "vs", the marginal facet for "vs" is also
# a margin over "am".
mg + facet_grid(vs + am ~ gear, margins = "vs")
mg + facet_grid(vs + am ~ gear, margins = "gear")
mg + facet_grid(vs + am ~ gear, margins = c("gear", "am"))

---

### facet_wrap

Wrap a 1d ribbon of panels into 2d

#### Description

facet_wrap wraps a 1d sequence of panels into 2d. This is generally a better use of screen space than facet_grid because most displays are roughly rectangular.

#### Usage

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

#### Arguments

- **facets**: Either a formula or character vector. Use either a one sided formula, \(\sim a + b\), 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 \(\sim\text{cyl} + \text{am}\). 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 labeler(). 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.
**facet_wrap**

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".

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.

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

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**

```r
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  facet_wrap(~class)

# Control the number of rows and columns with nrow and ncol
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  facet_wrap(~class, nrow = 4)

# You can facet by multiple variables
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  facet_wrap(~cyl + drv)
# Or use a character vector:
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  facet_wrap(c("cyl", "drv"))

# Use the 'labeller' option to control how labels are printed:
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  facet_wrap(c("cyl", "drv"), labeller = "label_both")

# To change the order in which the panels appear, change the levels
# of the underlying factor.
mpg$class2 <- reorder(mpg$class, mpg$displ)
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  facet_wrap(~class2)

# By default, the same scales are used for all panels. You can allow
# scales to vary across the panels with the 'scales' argument.
# Free scales make it easier to see patterns within each panel, but
# harder to compare across panels.
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  facet_wrap(~class, scales = "free")
```
To repeat the same data in every panel, simply construct a data frame that does not contain the facetting variable.

```r
ggplot(mpg, aes(displ, hwy)) +
  geom_point(data = transform(mpg, class = NULL), colour = "grey85") +
  geom_point() +
  facet_wrap(~class)
```

Use `strip.position` to display the facet labels at the side of your choice. Setting it to `"bottom"` makes it act as a subtitle for the axis. This is typically used with free scales and a theme without boxes around strip labels.

```r
ggplot(economics_long, aes(date, value)) +
  geom_line() +
  facet_wrap(~variable, scales = "free_y", nrow = 2, strip.position = "bottom") +
  theme(strip.background = element_blank(), strip.placement = "outside")
```

---

**faithful**

2d density estimate of Old Faithful data

**Description**

A 2d density estimate of the waiting and eruptions variables data `faithful`.

**Usage**

`faithful()`

**Format**

A data frame with 5,625 observations and 3 variables.

---

**fortify**

Fortify a model with data.

**Description**

Rather than using this function, I now recommend using the `broom` package, which implements a much wider range of methods. `fortify` may be deprecated in the future.

**Usage**

`fortify(model, data, ...)`
geom_abline

Arguments

model          model or other R object to convert to data frame
data           original dataset, if needed
...            other arguments passed to methods

See Also

fortify.lm

---

**geom_abline**

Reference lines: horizontal, vertical, and diagonal

---

Description

These geoms add reference lines (sometimes called rules) to a plot, either horizontal, vertical, or diagonal (specified by slope and intercept). These are useful for annotating plots.

Usage

```r
geom_abline(mapping = NULL, data = NULL, ..., slope, intercept, 
na.rm = FALSE, show.legend = NA)
geom_hline(mapping = NULL, data = NULL, ..., yintercept, na.rm = FALSE, 
show.legend = NA)
geom_vline(mapping = NULL, data = NULL, ..., xintercept, na.rm = FALSE, 
show.legend = NA)
```

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.

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

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.

xintercept, yintercept, slope, intercept
  Parameters that control the position of the line. If these are set, data, mapping and show.legend are overridden

Details

These geoms act slightly different to other geoms. You can supply the parameters in two ways: either as arguments to the layer function, or via aesthetics. If you use arguments, e.g. `geom_abline(intercept = 0, slope = ) then behind the scenes the geom makes a new data frame containing just the data you’ve supplied. That means that the lines will be the same in all facets; if you want them to vary across facets, construct the data frame yourself and use aesthetics.

Unlike most other geoms, these geoms do not inherit aesthetics from the plot default, because they do not understand x and y aesthetics which are commonly set in the plot. They also do not affect the x and y scales.

Aesthetics

These geoms are drawn using with `geom_line` so support the same aesthetics: alpha, colour, linetype and size. They also each have aesthetics that control the position of the line:

  • `geom_vline`: xintercept
  • `geom_hline`: yintercept
  • `geom_abline`: slope and intercept

See Also

See `geom_segment` for a more general approach to adding straight line segments to a plot.

Examples

```r
p <- ggplot(mtcars, aes(wt, mpg)) + geom_point()

# Fixed values
p + geom_vline(xintercept = 5)
p + geom_vline(xintercept = 1:5)
p + geom_hline(yintercept = 20)

p + geom_abline()  # Can't see it - outside the range of the data
p + geom_abline(intercept = 20)

# Calculate slope and intercept of line of best fit
coef(lm(mpg ~ wt, data = mtcars))
p + geom_abline(intercept = 37, slope = -5)
# But this is easier to do with geom_smooth:
p + geom_smooth(method = "lm", se = FALSE)
```
geom_bar

# To show different lines in different facets, use aesthetics
p <- ggplot(mtcars, aes(mpg, wt)) +
  geom_point() +
  facet_wrap(~ cyl)

mean_wt <- data.frame(cyl = c(4, 6, 8), wt = c(2.28, 3.11, 4.00))
p + geom_hline(aes(yintercept = wt), mean_wt)

# You can also control other aesthetics
ggplot(mtcars, aes(mpg, wt, colour = wt)) +
  geom_point() +
  geom_hline(aes(yintercept = wt, colour = wt), mean_wt) +
  facet_wrap(~ cyl)

---

gem_bar

Bars charts

Description

There are two types of bar charts: geom_bar makes the height of the bar proportional to the number of cases in each group (or if the weight aesthetic is supplied, the sum of the weights). If you want the heights of the bars to represent values in the data, use geom_col instead. geom_bar uses stat_count by default: it counts the number of cases at each x position. geom_col uses stat_identity: it leaves the data as is.

Usage

geom_bar(mapping = NULL, data = NULL, stat = "count",
         position = "stack", ..., width = NULL, binwidth = NULL, na.rm = FALSE,
         show.legend = NA, inherit.aes = TRUE)

geom_col(mapping = NULL, data = NULL, position = "stack", ...
         width = NULL, na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)

stat_count(mapping = NULL, data = NULL, geom = "bar",
           position = "stack", ..., width = 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.
geom_bar

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.

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 color = "red" or size = 3. They may also be parameters to the paired geom/stat.

width

Bar width. By default, set to 90% of the resolution of the data.

binwidth

geom_bar no longer has a binwidth argument - if you use it you’ll get an warning telling you use geom_histogram instead.

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.

inherit.aes

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

geom, stat

Override the default connection between geom_bar and stat_count.

Details

A bar chart uses height to represent a value, and so the base of the bar must always be shown to produce a valid visual comparison. Naomi Robbins has a nice article on this topic. This is why it doesn’t make sense to use a log-scaled y axis with a bar chart.

By default, multiple bar occupying the same x position will be stacked atop one another by position_stack. If you want them to be dodged side-to-side, use position_dodge. Finally, position_fill shows relative proportions at each x by stacking the bars and then standardising each bar to have the same height.

Aesthetics

geom_bar understands the following aesthetics (required aesthetics are in bold):

• x
• y
• alpha
• colour
• fill
• group
• linetype
• size
Computed variables

- **count**: number of points in bin
- **prop**: groupwise proportion

See Also

- `geom_histogram` for continuous data, `position_dodge` for creating side-by-side barcharts.
- `stat_bin`, which bins data in ranges and counts the cases in each range. It differs from `stat_count`, which counts the number of cases at each x position (without binning into ranges). `stat_bin` requires continuous x data, whereas `stat_count` can be used for both discrete and continuous x data.

Examples

```r
# geom_bar is designed to make it easy to create bar charts that show
# counts (or sums of weights)
g <- ggplot(mpg, aes(class))
# Number of cars in each class:
g + geom_bar()
# Total engine displacement of each class
g + geom_bar(aes(weight = displ))

# To show (e.g.) means, you need geom_col()
# And, even more succinctly with geom_col()
df <- data.frame(trt = c("a", "b", "c"), outcome = c(2.3, 1.9, 3.2))
ggplot(df, aes(trt, outcome)) + geom_col()
# But geom_point() displays exactly the same information and doesn't
# require the y-axis to touch zero.
ggplot(df, aes(trt, outcome)) + geom_point()

# You can also use geom_bar() with continuous data, in which case
# it will show counts at unique locations
df <- data.frame(x = rep(c(2.9, 3.1, 4.5), c(5, 10, 4)))
ggplot(df, aes(x)) + geom_bar()
# cf. a histogram of the same data
ggplot(df, aes(x)) + geom_histogram(binwidth = 0.5)

# Bar charts are automatically stacked when multiple bars are placed
# at the same location
g + geom_bar(aes(fill = drv))

# You can instead dodge, or fill them
g + geom_bar(aes(fill = drv), position = "dodge")
g + geom_bar(aes(fill = drv), position = "fill")

# To change plot order of bars, change levels in underlying factor
reorder_size <- function(x) {
  factor(x, levels = names(sort(table(x))))
}
geom_bin2d

Heatmap of 2d bin counts

Description

Divides the plane into rectangles, counts the number of cases in each rectangle, and then (by default) maps the number of cases to the rectangle's fill. This is a useful alternative to `geom_point` in the presence of overplotting.

Usage

```r
geom_bin2d(mapping = NULL, data = NULL, stat = "bin2d",
position = "identity", ..., na.rm = FALSE, show.legend = NA,
inherit.aes = TRUE)
```

```r
stat_bin_2d(mapping = NULL, data = NULL, geom = "tile",
position = "identity", ..., bins = 30, binwidth = NULL, drop = TRUE,
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.
- **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 `color = "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.
geom_blank

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>geom, stat</td>
<td>Use to override the default connection between <code>geom_bin2d</code> and <code>stat_bin2d</code>.</td>
</tr>
<tr>
<td>bins</td>
<td>numeric vector giving number of bins in both vertical and horizontal directions. Set to 30 by default.</td>
</tr>
<tr>
<td>binwidth</td>
<td>Numeric vector giving bin width in both vertical and horizontal directions. Overrides bins if both set.</td>
</tr>
<tr>
<td>drop</td>
<td>if TRUE removes all cells with 0 counts.</td>
</tr>
</tbody>
</table>

### Aesthetics

`stat_bin2d` understands the following aesthetics (required aesthetics are in bold):

- x
- y
- fill
- group

### See Also

- `stat_binhex` for hexagonal binning

### Examples

```r
d <- ggplot(diamonds, aes(x, y)) + xlim(4, 10) + ylim(4, 10)
d + geom_bin2d()
```

# You can control the size of the bins by specifying the number of bins in each direction:

d + geom_bin2d(bins = 10)
d + geom_bin2d(bins = 30)

# Or by specifying the width of the bins

d + geom_bin2d(binwidth = c(0.1, 0.1))
```

---

**geom_blank**

*Draw nothing*

### Description

The blank geom draws nothing, but can be a useful way of ensuring common scales between different plots. See `expand_limits` for more details.
Usage

    geom_blank(mapping = NULL, data = NULL, stat = "identity",
               position = "identity", ..., show.legend = NA, inherit.aes = TRUE)

Arguments

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

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

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

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

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

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.

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`.

Examples

    ggplot(mtcars, aes(wt, mpg))
    # Nothing to see here!

---

`geom_boxplot` A box and whiskers plot (in the style of Tukey)

Description

The boxplot compactly displays the distribution of a continuous variable. It visualises five summary
statistics (the median, two hinges and two whiskers), and all “outlying” points individually.
Usage

geom_boxplot(mapping = NULL, data = NULL, stat = "boxplot",
position = "$dodge$", ..., outlier.colour = NULL, outlier.color = NULL,
outlier.fill = NULL, outlier.shape = 19, outlier.size = 1.5,
outlier.stroke = 0.5, outlier.alpha = NULL, notch = FALSE,
notchwidth = 0.5, varwidth = FALSE, na.rm = FALSE, show.legend = NA,
inherit.aes = TRUE)

stat_boxplot(mapping = NULL, data = NULL, geom = "boxplot",
position = "$dodge$", ..., coef = 1.5, 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.

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 color = "red" or size = 3. They may also be parameters to the paired geom/stat.

outlier.colour, outlier.color, outlier.fill, outlier.shape, outlier.size, outlier.stroke, outlier.alpha Default aesthetics for outliers. Set to NULL to inherit from the aesthetics used for the box.
In the unlikely event you specify both US and UK spellings of colour, the US spelling will take precedence.

notch if FALSE (default) make a standard box plot. If TRUE, make a notched box plot. Notches are used to compare groups; if the notches of two boxes do not overlap, this suggests that the medians are significantly different.

notchwidth for a notched box plot, width of the notch relative to the body (default 0.5)

varwidth if FALSE (default) make a standard box plot. If TRUE, boxes are drawn with widths proportional to the square-roots of the number of observations in the groups (possibly weighted, using the weight aesthetic).

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.

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

geom, stat Use to override the default connection between geom_boxplot and stat_boxplot.

coeff length of the whiskers as multiple of IQR. Defaults to 1.5

Summary statistics

The lower and upper 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.

The upper whisker extends from the hinge to the largest value no further than 1.5 * IQR from the hinge (where IQR is the inter-quartile range, or distance between the first and third quartiles). The lower whisker extends from the hinge to the smallest value at most 1.5 * IQR of the hinge. Data beyond the end of the whiskers are called “outlying” points and are plotted individually.

In a notched box plot, the notches extend \(1.58 \times \text{IQR} / \sqrt{n}\). This gives a roughly 95% confidence interval for comparing medians. See McGill et al. (1978) for more details.

Aesthetics

gem_boxplot understands the following aesthetics (required aesthetics are in bold):

- x
- lower
- upper
- middle
- ymin
- ymax
- alpha
- colour
- fill
- group
- linetype
- shape
- size
- weight
Computed variables

- **width**  width of boxplot
- **ymin**  lower whisker = smallest observation greater than or equal to lower hinge - 1.5 * IQR
- **lower**  lower hinge, 25% quantile
- **notchlower**  lower edge of notch = median - 1.58 * IQR / sqrt(n)
- **middle**  median, 50% quantile
- **notchupper**  upper edge of notch = median + 1.58 * IQR / sqrt(n)
- **upper**  upper hinge, 75% quantile
- **ymax**  upper whisker = largest observation less than or equal to upper hinge + 1.5 * IQR

References


See Also

- `geom_quantile` for continuous x, `geom_violin` for a richer display of the distribution, and `geom_jitter` for a useful technique for small data.

Examples

```r
p <- ggplot(mpg, aes(class, hwy))
p + geom_boxplot()
p + geom_boxplot() + geom_jitter(width = 0.2)
p + geom_boxplot() + coord_flip()

p + geom_boxplot(notch = TRUE)
p + geom_boxplot(varwidth = TRUE)
p + geom_boxplot(fill = "white", colour = "#3366FF")
# By default, outlier points match the colour of the box. Use
# outlier.colour to override
p + geom_boxplot(outlier.colour = "red", outlier.shape = 1)

# Boxplots are automatically dodged when any aesthetic is a factor
p + geom_boxplot(aes(colour = drv))

# You can also use boxplots with continuous x, as long as you supply
# a grouping variable. cut_width is particularly useful
ggplot(diamonds, aes(carat, price)) +
  geom_boxplot()
ggplot(diamonds, aes(carat, price)) +
  geom_boxplot(aes(group = cut_width(carat, 0.25)))
ggplot(diamonds, aes(carat, price)) +
  geom_boxplot(aes(group = cut_width(carat, 0.25)), outlier.alpha = 0.1)

# It's possible to draw a boxplot with your own computations if you
# use stat = "identity":
```
geom_contour

2d contours of a 3d surface

Description

ggplot2 can not draw true 3d surfaces, but you can use geom_contour and geom_tile to visualise 3d surfaces in 2d. To be a valid surface, the data must contain only a single row for each unique combination of the variables mapped to the x and y aesthetics. Contouring tends to work best when x and y form a (roughly) evenly spaced grid. If your data is not evenly spaced, you may want to interpolate to a grid before visualising.

Usage

geom_contour(mapping = NULL, data = NULL, stat = "contour",
position = "identity", ..., lineend = "butt", linejoin = "round",
linemiter = 1, na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)

stat_contour(mapping = NULL, data = NULL, geom = "contour",
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.

**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 `color = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

**lineend**

Line end style (round, butt, square)

**linejoin**

Line join style (round, mitre, bevel)

**linemitre**

Line mitre limit (number greater than 1)

**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.

**inherit.aes**

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

**geom**

The geometric object to use display the data

### Aesthetics

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

- x
- y
- alpha
- colour
- group
- linetype
- size
- weight

### Computed variables

- level height of contour

### See Also

- `geom_density_2d`: 2d density contours
Examples

```r
# Basic plot
v <- ggplot(faithful, aes(waiting, eruptions, z = density))
v + geom_contour()

# Or compute from raw data
ggplot(faithful, aes(waiting, eruptions)) +
  geom_density_2d()

# Setting bins creates evenly spaced contours in the range of the data
v + geom_contour(bins = 2)
v + geom_contour(bins = 10)

# Setting binwidth does the same thing, parameterised by the distance
# between contours
v + geom_contour(binwidth = 0.01)
v + geom_contour(binwidth = 0.001)

# Other parameters
v + geom_contour(aes(colour = ..level..))
v + geom_contour(colour = "red")
v + geom_raster(aes(fill = density)) +
  geom_contour(colour = "white")
```

---

**geom_count**  
*Count overlapping points*

Description

This is a variant of `geom_point` that counts the number of observations at each location, then maps the count to point area. It is useful when you have discrete data and overplotting.

Usage

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

```r
stat_sum(mapping = NULL, data = NULL, geom = "point",
  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.
**geom_count**

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.

**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 color = "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.

**inherit.aes**

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

**geom, stat**

Use to override the default connection between `geom_count` and `stat_sum`.

**Aesthetics**

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

- x
- y
- alpha
- colour
- fill
- group
- shape
- size
- stroke

**Computed variables**

- n number of observations at position
- prop percent of points in that panel at that position

**See Also**

For continuous x and y, use `geom_bin2d`. 
Examples

```r
ggplot(mpg, aes(cty, hwy)) + geom_point()

ggplot(mpg, aes(cty, hwy)) + geom_count()

# Best used in conjunction with scale_size_area which ensures that counts of zero would be given size 0. Doesn't make much different here because the smallest count is already close to 0.

ggplot(mpg, aes(cty, hwy)) + geom_count() + scale_size_area()

# Display proportions instead of counts -------------------------------
# By default, all categorical variables in the plot form the groups.
# Specifying geom_count without a group identifier leads to a plot which is not useful.

d <- ggplot(diamonds, aes(x = cut, y = clarity))
d + geom_count(aes(size = ..prop..))

# To correct this problem and achieve a more desirable plot, we need to specify which group the proportion is to be calculated over.
d + geom_count(aes(size = ..prop.., group = 1)) + scale_size_area(max_size = 10)

# Or group by x/y variables to have rows/columns sum to 1.
d + geom_count(aes(size = ..prop.., group = cut)) + scale_size_area(max_size = 10)
d + geom_count(aes(size = ..prop.., group = clarity)) + scale_size_area(max_size = 10)
```

---

### geom_crossbar

**Vertical intervals: lines, crossbars & errorbars**

**Description**

Various ways of representing a vertical interval defined by x, ymin and ymax. Each case draws a single graphical object.

**Usage**

```r
geom_crossbar(mapping = NULL, data = NULL, stat = "identity", position = "identity", ..., fatten = 2.5, na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)

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

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

ggeom_pointrange(mapping = NULL, data = NULL, stat = "identity", 
position = "identity", ..., fatten = 4, 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.
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 color = "red" or size = 3. They may also be parameters to the paired geom/stat.
fatten A multiplicative factor used to increase the size of the middle bar in geom_crossbar() and the middle point in geom_pointrange().
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.
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

ggeom_linerange understands the following aesthetics (required aesthetics are in bold):

• x
• ymin
• ymax
• alpha
• colour
• group
• linetype
• size

See Also

stat_summary for examples of these guys in use, geom_smooth for continuous analog, geom_errorbarh for a horizontal error bar.

Examples

```r
# Create a simple example dataset
df <- data.frame(
  trt = factor(c(1, 1, 2, 2)),
  resp = c(1, 5, 3, 4),
  group = factor(c(1, 2, 1, 2)),
  upper = c(1.1, 5.3, 3.3, 4.2),
  lower = c(0.8, 4.6, 2.4, 3.6)
)
p <- ggplot(df, aes(trt, resp, colour = group))
  p + geom_linerange(aes(ymin = lower, ymax = upper))
p + geom_pointrange(aes(ymin = lower, ymax = upper))
p + geom_crossbar(aes(ymin = lower, ymax = upper), width = 0.2)
p + geom_errorbar(aes(ymin = lower, ymax = upper), width = 0.2)

# Draw lines connecting group means
p +
  geom_line(aes(group = group)) +
  geom_errorbar(aes(ymin = lower, ymax = upper), width = 0.2)

# If you want to dodge bars and errorbars, you need to manually
# specify the dodge width
p <- ggplot(df, aes(trt, resp, fill = group))
p +
  geom_col(position = "dodge") +
  geom_errorbar(aes(ymin = lower, ymax = upper), position = "dodge", width = 0.25)

# Because the bars and errorbars have different widths
# we need to specify how wide the objects we are dodging are
dodge <- position_dodge(width=0.9)
p +
  geom_col(position = dodge) +
  geom_errorbar(aes(ymin = lower, ymax = upper), position = dodge, width = 0.25)
```
**geom_density**

Smoothed density estimates

**Description**

Computes and draws kernel density estimate, which is a smoothed version of the histogram. This is a useful alternative to the histogram if for continuous data that comes from an underlying smooth distribution.

**Usage**

```r
geom_density(mapping = NULL, data = NULL, stat = "density",
            position = "identity", ..., na.rm = FALSE, show.legend = NA,
            inherit.aes = true)
```

```r
stat_density(mapping = NULL, data = NULL, geom = "area",
            position = "stack", ..., bw = "nrd0", adjust = 1, kernel = "gaussian",
            n = 512, trim = FALSE, 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.

- **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 `color = "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.

- **inherit.aes**: If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders`.
geom, stat
Use to override the default connection between geom_density and stat_density.

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 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.

Aesthetics

geom_density understands the following aesthetics (required aesthetics are in bold):

- x
- y
- alpha
- colour
- fill
- group
- linetype
- size
- weight

Computed variables

density density estimate
count density * number of points - useful for stacked density plots
scaled density estimate, scaled to maximum of 1

See Also

See geom_histogram, geom_freqpoly for other methods of displaying continuous distribution. See geom_violin for a compact density display.
Examples

```r
# Stacked density plots: if you want to create a stacked density plot, you
# probably want to 'count' (density * n) variable instead of the default
# density

# Loses marginal densities
ggplot(diamonds, aes(carat, fill = cut)) +
  geom_density(position = "stack")
# Preserves marginal densities
ggplot(diamonds, aes(carat, ..count.., fill = cut)) +
  geom_density(position = "stack")

# You can use position="fill" to produce a conditional density estimate
ggplot(diamonds, aes(carat, ..count.., fill = cut)) +
  geom_density(position = "fill")
```

---

**geom_density_2d**

Contours of a 2d density estimate

Description

Perform a 2D kernel density estimation using `kde2d` and display the results with contours. This can be useful for dealing with overplotting. This is a 2d version of `geom_density`.

Usage

```r
geom_density_2d(mapping = NULL, data = NULL, stat = "density2d",
    position = "identity", ..., lineend = "butt", linejoin = "round",
    linemitre = 1, na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```

```r
stat_density_2d(mapping = NULL, data = NULL, geom = "density_2d",
    position = "identity", ..., contour = TRUE, n = 100, h = 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.

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 color = "red" or size = 3. They may also be parameters to the paired geom/stat.

lineend Line end style (round, butt, square)
linestyle Line join style (round, mitre, bevel)
linemitre Line mitre limit (number greater than 1)

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.

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

geom, stat Use to override the default connection between geom_density_2d and stat_density_2d.
contour If TRUE, contour the results of the 2d density estimation

n number of grid points in each direction
h Bandwidth (vector of length two). If NULL, estimated using bandwidth.nrd.

Aesthetics

geom_density_2d understands the following aesthetics (required aesthetics are in bold):

- x
- y
- alpha
- colour
- group
- linetype
- size
Computation variables

Same as `stat_contour`

See Also

`geom_contour` for information about how contours are drawn; `geom_bin2d` for another way of dealing with overplotting.

Examples

```r
m <- ggplot(faithful, aes(x = eruptions, y = waiting)) +
  geom_point() +
  xlim(0.5, 6) +
  ylim(40, 110)

m + geom_density_2d()

m + stat_density_2d(aes(fill = ..level..), geom = "polygon")

set.seed(4393)
dsmall <- diamonds[sample(nrow(diamonds), 1000),]
d <- ggplot(dsmall, aes(x, y))
# If you map an aesthetic to a categorical variable, you will get a
# set of contours for each value of that variable

m + geom_density_2d(aes(colour = cut))

# If we turn contouring off, we can use use geoms like tiles:
d + stat_density_2d(geom = "raster", aes(fill = ..density..), contour = FALSE)
# Or points:
d + stat_density_2d(geom = "point", aes(size = ..density..), n = 20, contour = FALSE)
```

---

**Description**

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.

**Usage**

```r
geom_dotplot(mapping = NULL, data = NULL, position = "identity", ...,
  binwidth = NULL, binaxis = "x", method = "dotdensity",
  binpositions = "bygroup", stackdir = "up", stackratio = 1,
  dotsize = 1, stackgroups = FALSE, origin = NULL, right = TRUE,
  width = 0.9, drop = FALSE, 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.

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 `color = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

binwidth When method is "dotdensity", this specifies maximum bin width. When method is "histodot", this specifies bin width. Defaults to 1/30 of the range of the data

binaxis The axis to bin along, "x" (default) or "y"

method "dotdensity" (default) for dot-density binning, or "histodot" for fixed bin widths (like `stat_bin`)

binpositions When method is "dotdensity", "bygroup" (default) determines positions of the bins for each group separately. "all" determines positions of the bins with all the data taken together; this is used for aligning dot stacks across multiple groups.

stackdir which direction to stack the dots. "up" (default), "down", "center", "centerwhole" (centered, but with dots aligned)

stackratio how close to stack the dots. Default is 1, where dots just just touch. Use smaller values for closer, overlapping dots.

dotsizesize The diameter of the dots relative to `binwidth`, default 1.

stackgroups should dots be stacked across groups? This has the effect that `position = "stack"` should have, but can't (because this geom has some odd properties).

origin When method is "histodot", origin of first bin

right When method is "histodot", should intervals be closed on the right (a, b], or not [a, b)

width When binaxis is "y", the spacing of the dot stacks for dodging.

drop If `TRUE`, remove all bins with zero counts

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.

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


Details

There are two basic approaches: dot-density and histodot. With dot-density binning, the bin positions are determined by the data and binwidth, which is the maximum width of each bin. See Wilkinson (1999) for details on the dot-density binning algorithm. With histodot binning, the bins have fixed positions and fixed widths, much like a histogram.

When binning along the x axis and stacking along the y axis, the numbers on y axis are not meaningful, due to technical limitations of ggplot2. You can hide the y axis, as in one of the examples, or manually scale it to match the number of dots.

Aesthetics

geom_dotplot understands the following aesthetics (required aesthetics are in bold):

- x
- y
- alpha
- colour
- fill
- group

Computed variables

- x  center of each bin, if binaxis is "x"
- y  center of each bin, if binaxis is "x"
- binwidth  max width of each bin if method is "dotdensity"; width of each bin if method is "histodot"
- count  number of points in bin
- ncount  count, scaled to maximum of 1
- density  density of points in bin, scaled to integrate to 1, if method is "histodot"
- ndensity  density, scaled to maximum of 1, if method is "histodot"

References


Examples

```r
ggplot(mtcars, aes(x = mpg)) + geom_dotplot()
```

# Use fixed-width bins
```
ggplot(mtcars, aes(x = mpg)) + geom_dotplot(binwidth = 1.5)
```

# Some other stacking methods
```
ggplot(mtcars, aes(x = mpg)) +
  geom_dotplot(binwidth = 1.5, stackdir = "center")
```
Horizontal error bars

Description

A rotated version of `geom_errorbar`.
Usage

geom_errorbarh(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.

- **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 `color = "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.

- **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_errorbarh` understands the following aesthetics (required aesthetics are in bold):

- **x**
- **xmin**
- **xmax**
- **y**
- **alpha**
- **colour**
- **group**
- **height**
- **linetype**
- **size**
Examples

df <- data.frame(
  trt = factor(c(1, 1, 2, 2)),
  resp = c(1, 5, 3, 4),
  group = factor(c(1, 2, 1, 2)),
  se = c(0.1, 0.3, 0.3, 0.2)
)

# Define the top and bottom of the errorbars
p <- ggplot(df, aes(resp, trt, colour = group))
p + geom_point() +
  geom_errorbarh(aes(xmax = resp + se, xmin = resp - se))
p + geom_point() +
  geom_errorbarh(aes(xmax = resp + se, xmin = resp - se, height = .2))

---

geom_freqpoly  Histories and frequency polygons

Description

Visualise the distribution of a single continuous variable by dividing the x axis into bins and counting the number of observations in each bin. Histograms (geom_histogram) display the count with bars; frequency polygons (geom_freqpoly), display the counts with lines. Frequency polygons are more suitable when you want to compare the distribution across a the levels of a categorical variable.

stat_bin is suitable only for continuous x data. If your x data is discrete, you probably want to use stat_count.

Usage

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

gem_histogram(mapping = NULL, data = NULL, stat = "bin",
  position = "stack", ..., binwidth = NULL, bins = NULL, na.rm = FALSE,
  show.legend = NA, inherit.aes = TRUE)

stat_bin(mapping = NULL, data = NULL, geom = "bar",
  position = "stack", 
  ..., binwidth = NULL, bins = NULL, center = NULL, boundary = NULL,
  breaks = NULL, closed = c("right", "left"), pad = FALSE,
  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.

**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 `color = "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.

**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`.

**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.
The bin width of a date variable is the number of days in each time; the bin width of a time variable is the number of seconds.

**bins**
Number of bins. Overridden by `binwidth`. Defaults to 30

**geom, stat**
Use to override the default connection between `geom_histogram/geom_freqpoly` and `stat_bin`.

**center**
The center of one of the bins. Note that if center is above or below the range of the data, things will be shifted by an appropriate number of widths. To center on integers, for example, use `width = 1` and `center = 0`, even if 0 is outside the range of the data. At most one of center and boundary may be specified.

**boundary**
A boundary between two bins. As with center, things are shifted when boundary is outside the range of the data. For example, to center on integers, use `width = 1` and `boundary = 0.5`, even if 0.5 is outside the range of the data. At most one of center and boundary may be specified.

**breaks**
Alternatively, you can supply a numeric vector giving the bin boundaries. Over-rides `binwidth`, `bins`, `center`, and `boundary`.

**closed**
One of "right" or "left" indicating whether right or left edges of bins are included in the bin.
pad If TRUE, adds empty bins at either end of x. This ensures frequency polygons touch 0. Defaults to FALSE.

Details

By default, the underlying computation (stat_bin) uses 30 bins - this is not a good default, but the idea is to get you experimenting with different binwidths. You may need to look at a few to uncover the full story behind your data.

Aesthetics

geom_histogram uses the same aesthetics as geom_bar; geom_freqpoly uses the same aesthetics as geom_line.

Computed variables

- **count**: number of points in bin
- **density**: density of points in bin, scaled to integrate to 1
- **ncount**: count, scaled to maximum of 1
- **ndensity**: density, scaled to maximum of 1

See Also

- **stat_count**, which counts the number of cases at each x position, without binning. It is suitable for both discrete and continuous x data, whereas **stat_bin** is suitable only for continuous x data.

Examples

```r
ggplot(diamonds, aes(carat)) + geom_histogram()

# Rather than stacking histograms, it's easier to compare frequency polygons
ggplot(diamonds, aes(price, fill = cut)) + geom_histogram(binwidth = 0.01)

# To make it easier to compare distributions with very different counts,
# put density on the y axis instead of the default count
ggplot(diamonds, aes(price, ..density.., colour = cut)) + geom_freqpoly(binwidth = 500)

if (require("ggplot2movies")) {
  # Often we don't want the height of the bar to represent the count of observations, but the sum of some other variable.
  # For example, the following plot shows the number of movies
```
# in each rating.
m <- ggplot(movies, aes(rating))
m + geom_histogram(binwidth = 0.1)

# If, however, we want to see the number of votes cast in each
category, we need to weight by the votes variable
m + geom_histogram(aes(weight = votes), binwidth = 0.1) + ylab("votes")

# For transformed scales, binwidth applies to the transformed data.
The bins have constant width on the transformed scale.
m + geom_histogram() + scale_x_log10()
m + geom_histogram(binwidth = 0.05) + scale_x_log10()

# For transformed coordinate systems, the binwidth applies to the
# raw data. The bins have constant width on the original scale.

# Using log scales does not work here, because the first
# bar is anchored at zero, and so when transformed becomes negative
# infinity. This is not a problem when transforming the scales, because
# no observations have 0 ratings.
m + geom_histogram(boundary = 0) + coord_trans(x = "log10")
# Use boundary = 0, to make sure we don’t take sqrt of negative values
m + geom_histogram(boundary = 0) + coord_trans(x = "sqrt")

# You can also transform the y axis. Remember that the base of the bars
# has value 0, so log transformations are not appropriate
m <- ggplot(movies, aes(x = rating))
m + geom_histogram(binwidth = 0.5) + scale_y_sqrt()

}

geom_hex

Hexagonal heatmap of 2d bin counts

Description

Divides the plane into regular hexagons, counts the number of cases in each hexagon, and then
(by default) maps the number of cases to the hexagon fill. Hexagon bins avoid the visual artefacts
sometimes generated by the very regular alignment of geom_bin2d.

Usage

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

stat_bin_hex(mapping = NULL, data = NULL, geom = "hex",
    position = "identity", ..., bins = 30, binwidth = 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.

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 `color = "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.

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

geom, stat Override the default connection between `geom_hex` and `stat_binhex`.

bins numeric vector giving number of bins in both vertical and horizontal directions. Set to 30 by default.

binwidth Numeric vector giving bin width in both vertical and horizontal directions. Overrides bins if both set.

Aesthetics

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

- `x`
- `y`
- `alpha`
- `colour`
- `fill`
- `group`
- `size`
See Also

stat_bin2d for rectangular binning

Examples

```r
d <- ggplot(diamonds, aes(carat, price))
d + geom_hex()

# You can control the size of the bins by specifying the number of
# bins in each direction:
d + geom_hex(bins = 10)
d + geom_hex(bins = 30)

# Or by specifying the width of the bins

d + geom_hex(binwidth = c(1, 1000))
d + geom_hex(binwidth = c(.1, 500))
```

geom_jitter

Jittered points

Description

The jitter geom is a convenient shortcut for geom_point(position = "jitter"). It adds a small amount of random variation to the location of each point, and is a useful way of handling overplotting caused by discreteness in smaller datasets.

Usage

```r
geom_jitter(mapping = NULL, data = NULL, stat = "identity",
position = "jitter", ..., width = NULL, height = 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.
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 color = "red" or size = 3. They may also be parameters to the paired geom/stat.

width
Amount of vertical and horizontal jitter. The jitter is added in both positive and negative directions, so the total spread is twice the value specified here. If omitted, defaults to 40% of the resolution of the data: this means the jitter values will occupy 80% of the implied bins. Categorical data is aligned on the integers, so a width or height of 0.5 will spread the data so it’s not possible to see the distinction between the categories.

height
Amount of vertical and horizontal jitter. The jitter is added in both positive and negative directions, so the total spread is twice the value specified here. If omitted, defaults to 40% of the resolution of the data: this means the jitter values will occupy 80% of the implied bins. Categorical data is aligned on the integers, so a width or height of 0.5 will spread the data so it’s not possible to see the distinction between the categories.

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.

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_point understands the following aesthetics (required aesthetics are in bold):

• x
• y
• alpha
• colour
• fill
• group
• shape
• size
• stroke

See Also

geom_point for regular, unjittered points, geom_boxplot for another way of looking at the conditional distribution of a variable
Examples

```r
p <- ggplot(mpg, aes(cyl, hwy))
p + geom_point()
p + geom_jitter()

# Add aesthetic mappings
p + geom_jitter(aes(colour = class))

# Use smaller width/height to emphasise categories
ggplot(mpg, aes(cyl, hwy)) + geom_jitter()

# Use larger width/height to completely smooth away discreteness
ggplot(mpg, aes(cyl, hwy)) + geom_jitter() + geom_jitter(width = 0.5, height = 0.5)
```

Description

gem_text adds text directly to the plot. geom_label draws a rectangle behind the text, making it easier to read.

Usage

```r
geom_label(mapping = NULL, data = NULL, stat = "identity",
position = "identity", ..., parse = FALSE, nudge_x = 0, nudge_y = 0,
lablel_padding = unit(0.25, "lines"), label.r = unit(0.15, "lines"),
lablel.size = 0.25, na.rm = FALSE, show.legend = NA,
inherit.aes = TRUE)

geom_text(mapping = NULL, data = NULL, stat = "identity",
position = "identity", ..., parse = FALSE, nudge_x = 0, nudge_y = 0,
check_overlap = FALSE, na.rm = FALSE, show.legend = NA,
inherite.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.

**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 `color = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

**parse**  
If TRUE, the labels will be parsed into expressions and displayed as described in `?plotmath`.

**nudge_x**, **nudge_y**  
Horizontal and vertical adjustment to nudge labels by. Useful for offsetting text from points, particularly on discrete scales.

**label.padding**  
Amount of padding around label. Defaults to 0.25 lines.

**label.r**  
Radius of rounded corners. Defaults to 0.15 lines.

**label.size**  
Size of label border, in mm.

**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.

**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`.

**check_overlap**  
If TRUE, text that overlaps previous text in the same layer will not be plotted.

**Details**

Note the the "width" and "height" of a text element are 0, so stacking and dodging text will not work by default, and axis limits are not automatically expanded to include all text. Obviously, labels do have height and width, but they are physical units, not data units. The amount of space they occupy on that plot is not constant in data units: when you resize a plot, labels stay the same size, but the size of the axes changes.

**Aesthetics**

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

- `x`
- `y`
- `label`
- `alpha`
- `angle`
- `colour`
- `family`
• fontface
• group
• hjust
• lineheight
• size
• vjust

geom_label

Currently geom_label does not support the rot parameter and is considerably slower than geom_text. The fill aesthetic controls the background colour of the label.

Alignment

You can modify text alignment with the vjust and hjust aesthetics. These can either be a number between 0 (right/bottom) and 1 (top/left) or a character ("left", "middle", "right", "bottom", "center", "top"). There are two special alignments: "inward" and "outward". Inward always aligns text towards the center, and outward aligns it away from the center.

Examples

p <- ggplot(mtcars, aes(wt, mpg, label = rownames(mtcars)))

p + geom_text()
  # Avoid overlaps
p + geom_text(check_overlap = TRUE)
  # Labels with background
p + geom_label()
  # Change size of the label
p + geom_text(size = 10)

  # Set aesthetics to fixed value
p + geom_point() + geom_text(hjust = 0, nudge_x = 0.05)
p + geom_point() + geom_text(vjust = 0, nudge_y = 0.5)
p + geom_point() + geom_text(angle = 45)

## Not run:
  # Doesn't work on all systems
p + geom_text(family = "Times New Roman")

  ## End(Not run)

  # Add aesthetic mappings
p + geom_text(aes(colour = factor(cyl)))
p + geom_text(aes(colour = factor(cyl))) +
  scale_colour_discrete(l = 40)
p + geom_label(aes(fill = factor(cyl)), colour = "white", fontface = "bold")

p + geom_text(aes(size = wt))
  # Scale height of text, rather than sqrt(height)
p + geom_text(aes(size = wt)) + scale_radius(range = c(3, 6))
# You can display expressions by setting parse = TRUE. The
# details of the display are described in ?plotmath, but note that
# geom_text uses strings, not expressions.
p + geom_text(aes(label = paste(wt, "^\(\cdot\)\(cyl\)\), sep = ""),
              parse = TRUE)

# Add a text annotation
p +
  geom_text(label = "plot mpg vs. wt", x = 2, y = 15, size = 8, colour = "red")

# Aligning labels and bars ------------------------------------------
df <- data.frame(
  x = factor(c(1, 1, 2, 2)),
  y = c(1, 3, 2, 1),
  grp = c("a", "b", "a", "b")
)

# ggplot2 doesn't know you want to give the labels the same virtual width
# as the bars:
ggplot(data = df, aes(x, y, group = grp)) +
  geom_col(aes(fill = grp), position = "dodge") +
  geom_text(aes(label = y), position = "dodge")
# So tell it:
ggplot(data = df, aes(x, y, group = grp)) +
  geom_col(aes(fill = grp), position = "dodge") +
  geom_text(aes(label = y), position = position_dodge(0.9))
# Use you can't nudge and dodge text, so instead adjust the y position

# To place text in the middle of each bar in a stacked barplot, you
# need to set the vjust parameter of position_stack()
ggplot(data = df, aes(x, y, group = grp)) +
  geom_col(aes(fill = grp)) +
  geom_text(aes(label = y), position = position_stack(vjust = 0.5))

# Justification -----------------------------------------------------------
df <- data.frame(
  x = c(1, 1, 2, 2, 1.5),
  y = c(1, 2, 1, 2, 1.5),
  text = c("bottom-left", "bottom-right", "top-left", "top-right", "center")
)
ggplot(df, aes(x, y)) +
  geom_text(aes(label = text))
ggplot(df, aes(x, y)) +
**geom_map**

```r
geom_text(aes(label = text), vjust = "inward", hjust = "inward")
```

---

**geom_map**  
Polygons from a reference map

---

**Description**

This is pure annotation, so does not affect position scales.

**Usage**

```r
geom_map(mapping = NULL, data = NULL, stat = "identity", ..., map,  
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.

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

- **map**
  
  Data frame that contains the map coordinates. This will typically be created using `fortify` on a spatial object. It must contain columns `x` or `long`, `y` or `lat`, and `region` or `id`.

- **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.

- **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_map understands the following aesthetics (required aesthetics are in bold):

- map_id
- alpha
- colour
- fill
- group
- linetype
- size

Examples

# When using geom_polygon, you will typically need two data frames:
# one contains the coordinates of each polygon (positions), and the
# other the values associated with each polygon (values). An id
# variable links the two together

gids <- factor(c("1.1", "2.1", "1.2", "2.2", "1.3", "2.3"))

gvalues <- data.frame(
  id = gids,
  value = c(3, 3.1, 3.1, 3.2, 3.15, 3.5)
)

gpositions <- data.frame(
  id = rep(gids, each = 4),
  x = c(2, 1, 1.1, 2.2, 1, 0, 0.3, 1.1, 2.2, 1.1, 1.2, 2.5, 1.1, 0.3,
        0.5, 1.2, 2.5, 1.2, 1.3, 2.7, 1.2, 0.5, 0.6, 1.3),
  y = c(-0.5, 0, 1, 0.5, 0, 0.5, 1.5, 1, 0.5, 1, 2.1, 1.7, 1, 1.5,
        2.2, 2.1, 1.7, 2.1, 3.2, 2.8, 2.1, 2.2, 3.3, 3.2)
)

ggplot(gvalues) +
  geom_map(aes(map_id = id), map = gpositions) +
  expand_limits(gpositions)

ggplot(gvalues, aes(fill = value)) +
  geom_map(aes(map_id = id), map = gpositions) +
  expand_limits(gpositions)

ggplot(gvalues, aes(fill = value)) +
  geom_map(aes(map_id = id), map = gpositions) +
  expand_limits(gpositions) + ylim(0, 3)

# Better example

crimes <- data.frame(state = tolower(rownames(USArests)), USArests)

crimesm <- reshape2::melt(crimes, id = 1)

if (require(maps)) {
  states_map <- map_data("state")
  gplot(crimms, aes(map_id = state)) +
    geom_map(aes(fill = Murder), map = states_map) +
}
geom_path

```
expand_limits(x = states_map$long, y = states_map$lat)
last_plot() + coord_map()
ggplot(crimesm, aes(map_id = state)) +
  geom_map(aes(fill = value), map = states_map) +
  expand_limits(x = states_map$long, y = states_map$lat) +
  facet_wrap(~ variable)
```

---

### geom_path

**Connect observations**

**Description**

`geom_path()` connects the observations in the order in which they appear in the data. `geom_line()` connects them in order of the variable on the x axis. `geom_step()` creates a stairstep plot, highlighting exactly when changes occur. The group aesthetic determines which cases are connected together.

**Usage**

```
geom_path(mapping = NULL, data = NULL, stat = "identity",
          position = "identity", ..., lineend = "butt", linejoin = "round",
          linemitre = 1, arrow = NULL, na.rm = FALSE, show.legend = NA,
          inherit.aes = TRUE)
```

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

```
geom_step(mapping = NULL, data = NULL, stat = "identity",
          position = "identity", direction = "hv", 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.
stat
  The statistical transformation to use on the data for this layer, as a string.

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

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

lineend
  Line end style (round, butt, square)

linejoin
  Line join style (round, mitre, bevel)

linemitre
  Line mitre limit (number greater than 1)

arrow
  Arrow specification, as created by arrow

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.

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.

direction
  direction of stairs: 'vh' for vertical then horizontal, or 'hv' for horizontal then
  vertical

Details
An alternative parameterisation is geom_segment: each line corresponds to a single case which
provides the start and end coordinates.

Aesthetics
geom_path understands the following aesthetics (required aesthetics are in bold):

• x
• y
• alpha
• colour
• group
• linetype
• size

See Also
geom_polygon: Filled paths (polygons); geom_segment: Line segments
Examples

# geom_line() is suitable for time series
ggplot(economics, aes(date, unemploy)) + geom_line()
ggplot(economics_long, aes(date, value01, colour = variable)) + geom_line()

# geom_step() is useful when you want to highlight exactly when
# the y value changes
recent $<$- economics[economics$date$ > as.Date("2013-01-01"), ]
ggplot(recent, aes(date, unemploy)) + geom_line()
ggplot(recent, aes(date, unemploy)) + geom_step()

# geom_path lets you explore how two variables are related over time,
# e.g. unemployment and personal savings rate
m $<$- ggplot(economics, aes(unemploy/pop, psavert))
m + geom_path()
m + geom_path(aes(colour = as.numeric(date)))

# Changing parameters ---------------------------------------------
ggplot(economics, aes(date, unemploy)) +
  geom_line(colour = "red")

# Use the arrow parameter to add an arrow to the line
# See ?arrow for more details
c $<$- ggplot(economics, aes(x = date, y = pop))
c + geom_line(arrow = arrow())
c + geom_line(
  arrow = arrow(angle = 15, ends = "both", type = "closed")
)

# Control line join parameters
df $<$- data.frame(x = 1:3, y = c(4, 1, 9))
base $<$- ggplot(df, aes(x, y))
base + geom_path(size = 10)
base + geom_path(size = 10, lineend = "round")
base + geom_path(size = 10, linejoin = "mitre", lineend = "butt")

# NAs break the line. Use na.rm = T to suppress the warning message
df $<$- data.frame(
  x = 1:5,
  y1 = c(1, 2, 3, 4, NA),
  y2 = c(NA, 2, 3, 4, 5),
  y3 = c(1, 2, NA, 4, 5)
)
ggplot(df, aes(x, y1)) + geom_point() + geom_line()
ggplot(df, aes(x, y2)) + geom_point() + geom_line()
ggplot(df, aes(x, y3)) + geom_point() + geom_line()

# Setting line type vs colour/size
# Line type needs to be applied to a line as a whole, so it can
# not be used with colour or size that vary across a line
geom_point

x <- seq(0.01, .99, length.out = 100)
df <- data.frame(
  x = rep(x, 2),
  y = c(qlogis(x), 2 * qlogis(x)),
  group = rep(c("a", "b"), each = 100)
)
p <- ggplot(df, aes(x=x, y=y, group=group))
# These work
p + geom_line(linetype = 2)
p + geom_line(aes(colour = group), linetype = 2)
p + geom_line(aes(colour = x))
# But this doesn't
should_stop(p + geom_line(aes(colour = x), linetype=2))

Description

The point geom is used to create scatterplots. The scatterplot is most useful for displaying the relationship between two continuous variables. It can be used to compare one continuous and one categorical variable, or two categorical variables, but a variation like geom_jitter, geom_count, or geom_bin2d is usually more appropriate.

Usage

geom_point(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.

stat The statistical transformation to use on the data for this layer, as a string.
The `bubblechart` is a scatterplot with a third variable mapped to the size of points. There are no special names for scatterplots where another variable is mapped to point shape or colour, however.

**Overplotting**

The biggest potential problem with a scatterplot is overplotting: whenever you have more than a few points, points may be plotted on top of one another. This can severely distort the visual appearance of the plot. There is no one solution to this problem, but there are some techniques that can help. You can add additional information with `geom_smooth`, `geom_quantile` or `geom_density_2d`. If you have few unique x values, `geom_boxplot` may also be useful.

Alternatively, you can summarise the number of points at each location and display that in some way, using `geom_count`, `geom_hex`, or `geom_density2d`.

Another technique is to make the points transparent (e.g. `geom_point(alpha = 0.05)`) or very small (e.g. `geom_point(shape = ".")`).

**Aesthetics**

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

- `x`
- `y`
- `alpha`
- `colour`
- `fill`
- `group`
- `shape`
- `size`
- `stroke`
Examples

```r
p <- ggplot(mtcars, aes(wt, mpg))
p + geom_point()

# Add aesthetic mappings
p + geom_point(aes(colour = factor(cyl)))
p + geom_point(aes(shape = factor(cyl)))
p + geom_point(aes(size = qsec))

# Change scales
p + geom_point(aes(colour = cyl)) + scale_colour_gradient(low = "blue")
p + geom_point(aes(shape = factor(cyl))) + scale_shape(solid = FALSE)

# Set aesthetics to fixed value
ggplot(mtcars, aes(wt, mpg)) + geom_point(colour = "red", size = 3)

# Varying alpha is useful for large datasets
d <- ggplot(diamonds, aes(carat, price))
d + geom_point(alpha = 1/10)
d + geom_point(alpha = 1/20)
d + geom_point(alpha = 1/100)

# For shapes that have a border (like 21), you can colour the inside and
# outside separately. Use the stroke aesthetic to modify the width of the
# border
ggplot(mtcars, aes(wt, mpg)) +
  geom_point(shape = 21, colour = "black", fill = "white", size = 5, stroke = 5)

# You can create interesting shapes by layering multiple points of
# different sizes
p <- ggplot(mtcars, aes(mpg, wt, shape = factor(cyl)))
p + geom_point(aes(colour = factor(cyl)), size = 4) +
  geom_point(colour = "grey90", size = 1.5)
p + geom_point(colour = "black", size = 4.5) +
  geom_point(colour = "pink", size = 4) +
  geom_point(aes(shape = factor(cyl)))

# These extra layers don’t usually appear in the legend, but we can
# force their inclusion
p + geom_point(colour = "black", size = 4.5, show.legend = TRUE) +
  geom_point(colour = "pink", size = 4, show.legend = TRUE) +
  geom_point(aes(shape = factor(cyl)))

# geom_point warns when missing values have been dropped from the data set
# and not plotted, you can turn this off by setting na.rm = TRUE
mtcars2 <- transform(mtcars, mpg = ifelse(runif(32) < 0.2, NA, mpg))
ggplot(mtcars2, aes(wt, mpg)) + geom_point()
ggplot(mtcars2, aes(wt, mpg)) + geom_point(na.rm = TRUE)
```
**Description**

Polygons are very similar to paths (as drawn by `geom_path`) except that the start and end points are connected and the inside is coloured by `fill`. The `group` aesthetic determines which cases are connected together into a polygon.

**Usage**

```r
geom_polygon(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.

- `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 `color = "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.

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

- **x**
- **y**
- **alpha**
- **colour**
- **fill**
- **group**
- **linetype**
- **size**

**See Also**

- `geom_path` for an unfilled polygon, `geom_ribbon` for a polygon anchored on the x-axis

**Examples**

```r
# When using `geom_polygon`, you will typically need two data frames:
# one contains the coordinates of each polygon (positions), and the
# other the values associated with each polygon (values). An `id`
# variable links the two together

dataframe <- data.frame(id = ids,
                        value = c(3, 3.1, 3.1, 3.2, 3.15, 3.5)
)  

dataframe <- data.frame(id = rep(ids, each = 4),
                        x = c(2, 1, 1.1, 2.2, 1, 0, 0.3, 1.1, 2.2, 1.1, 1.2, 2.5, 1.1, 0.3,
                              0.5, 1.2, 2.5, 1.2, 1.3, 2.7, 1.2, 0.5, 0.6, 1.3),
                        y = c(-0.5, 0, 1, 0.5, 0, 0.5, 1.5, 1, 0.5, 1, 2.1, 1.7, 1, 1.5,
                              2.2, 2.1, 1.7, 2.1, 3.2, 2.8, 2.1, 2.2, 3.3, 3.2)
)  

# Currently we need to manually merge the two together
dataframe <- merge(values, positions, by = c("id"))

ggplot(dataframe, aes(x = x, y = y)) +
  geom_polygon(aes(fill = value, group = id))
```

# Which seems like a lot of work, but then it's easy to add on
# other features in this coordinate system, e.g.:

```r
stream <- data.frame(
```

```r
```
geom_qq

```r
x = cumsum(runif(50, max = 0.1)),
y = cumsum(runif(50,max = 0.1))
```

p + geom_line(data = stream, colour = "grey30", size = 5)

# And if the positions are in longitude and latitude, you can use
# coord_map to produce different map projections.

---

**geom_qq**  
*A quantile-quantile plot*

**Description**

A quantile-quantile plot

**Usage**

```r
geom_qq(mapping = NULL, data = NULL, geom = "point",
position = "identity", ..., distribution = stats::qnorm,
dparams = list(), na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```

```r
stat_qq(mapping = NULL, data = NULL, geom = "point",
position = "identity", ..., distribution = stats::qnorm,
dparams = list(), 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.

- **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.

- **distribution**
  - Distribution function to use, if `x` not specified
dparams Additional parameters passed on to distribution 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.

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

stat_qq understands the following aesthetics (required aesthetics are in bold):

- sample
- group
- x
- y

Computed variables

- sample sample quantiles
- theoretical theoretical quantiles

Examples

```r
df <- data.frame(y = rt(200, df = 5))
p <- ggplot(df, aes(sample = y))
p + stat_qq()
p + geom_point(stat = "qq")

# Use fitdistr from MASS to estimate distribution params
params <- as.list(MASS::fitdistr(df$y, "t")$estimate)
ggplot(df, aes(sample = y)) +
  stat_qq(distribution = qt, dparams = params["df"])

# Using to explore the distribution of a variable
ggplot(mtcars) +
  stat_qq(aes(sample = mpg))
ggplot(mtcars) +
  stat_qq(aes(sample = mpg, colour = factor(cyl)))
```
**geom_quantile**

**Quantile regression**

**Description**

This fits a quantile regression to the data and draws the fitted quantiles with lines. This is as a continuous analogue to **geom_boxplot**.

**Usage**

```r
geom_quantile(mapping = NULL, data = NULL, stat = "quantile",
position = "identity", ..., lineend = "butt", linejoin = "round",
linemitre = 1, na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```

```r
stat_quantile(mapping = NULL, data = NULL, geom = "quantile",
position = "identity", ..., quantiles = c(0.25, 0.5, 0.75),
formula = NULL, method = "rq", method.args = list(), 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.

- **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 `color = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

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

- **linejoin**
  Line join style (round, mitre, bevel)

- **linemitre**
  Line mitre limit (number greater than 1)

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

geom, stat  Use to override the default connection between geom_quantile and stat_quantile.

quantiles  conditional quantiles of y to calculate and display

formula  formula relating y variables to x variables

method  Quantile regression method to use. Currently only supports rq.

method.args  List of additional arguments passed on to the modelling function defined by
   method.

Aesthetics

geom_quantile understands the following aesthetics (required aesthetics are in bold):

• x
• y
• alpha
• colour
• group
• linetype
• size
• weight

Computed variables

quantile  quantile of distribution

Examples

m <- ggplot(mpg, aes(displ, 1/hwy)) + geom_point()
m + geom_quantile()  # You can also use rqss to fit smooth quantiles
m + geom_quantile(method = "rqss")  # Note that rqss doesn't pick a smoothing constant automatically, so
m + geom_quantile(method = "rqss", lambda = 0.1)  # you'll need to tweak lambda yourself

# Set aesthetics to fixed value
m + geom_quantile(colour = "red", size = 2, alpha = 0.5)
Description

`geom_rect` and `geom_tile` do the same thing, but are parameterised differently: `geom_rect` uses the locations of the four corners (xmin, xmax, ymin and ymax), while `geom_tile` uses the center of the tile and its size (x, y, width, height). `geom_raster` is a high performance special case for when all the tiles are the same size.

Usage

```r
geom_raster(mapping = NULL, data = NULL, stat = "identity",
position = "identity", ..., hjust = 0.5, vjust = 0.5,
interpolate = FALSE, na.rm = FALSE, show.legend = NA,
inherit.aes = TRUE)
```

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

```r
geom_tile(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.

- **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 `color = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.
**geom_raster**

- **hjust, vjust**: horizontal and vertical justification of the grob. Each justification value should be a number between 0 and 1. Defaults to 0.5 for both, centering each pixel over its data location.
- **interpolate**: If TRUE interpolate linearly, if FALSE (the default) don’t interpolate.
- **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.
- **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_tile` understands the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- **alpha**
- **colour**
- **fill**
- **group**
- **linetype**
- **size**

**Examples**

```r
# The most common use for rectangles is to draw a surface. You always want
to use geom_raster here because it's so much faster, and produces
# smaller output when saving to PDF
ggplot(faithful, aes(waiting, eruptions)) +
  geom_raster(aes(fill = density))

# Interpolation smooths the surface & is most helpful when rendering images.
ggplot(faithful, aes(waiting, eruptions)) +
  geom_raster(aes(fill = density), interpolate = TRUE)

# If you want to draw arbitrary rectangles, use geom_tile() or geom_rect()
df <- data.frame(
  x = rep(c(2, 5, 7, 9, 12), 2),
  y = rep(c(1, 2), each = 5),
  z = factor(rep(1:5, each = 2)),
  w = rep(diff(c(0, 4, 6, 8, 10, 14)), 2)
)
ggplot(df, aes(x, y)) +
  geom_tile(aes(fill = z))
ggplot(df, aes(x, y)) +
```
geom_tile(aes(fill = z, width = w), colour = "grey50")
ggplot(df, aes(xmin = x - w / 2, xmax = x + w / 2, ymin = y, ymax = y + 1)) +
  geom_rect(aes(fill = z, width = w), colour = "grey50")

# Justification controls where the cells are anchored
df <- expand.grid(x = 0:5, y = 0:5)
df$z <- runif(nrow(df))
# default is compatible with geom_tile()
ggplot(df, aes(x, y, fill = z)) + geom_raster()
# zero padding
  gggplot(df, aes(x, y, fill = z)) + geom_raster(hjust = 0, vjust = 0)

# Inspired by the image-density plots of Ken Knoblauch
cars <- ggrepplot(mtcars, aes(mpg, factor(cyl)))
cars + geom_point()
cars + stat_bin2d(aes(fill = ..count..), binwidth = c(3,1))
cars + stat_bin2d(aes(fill = ..density..), binwidth = c(3,1))

cars + stat_density(aes(fill = ..density..), geom = "raster", position = "identity")
cars + stat_density(aes(fill = ..count..), geom = "raster", position = "identity")

---

**geom_ribbon**  
**Ribbons and area plots**

**Description**

For each x value, geom_ribbon displays a y interval defined by ymin and ymax. geom_area is a special case of geom_ribbon, where the ymin is fixed to 0.

**Usage**

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

gm_area(mapping = NULL, data = NULL, stat = "identity",
         position = "stack", 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 ggpplot.
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.

`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 color = "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.

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

### Details

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. See `position_stack` for the details of stacking algorithm.

### Aesthetics

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

- x
- ymin
- ymax
- alpha
- colour
- fill
- group
- linetype
- size

### See Also

- `geom_bar` for discrete intervals (bars), `geom_linerange` for discrete intervals (lines), `geom_polygon` for general polygons
Examples

# Generate data
huron <- data.frame(year = 1875:1972, level = as.vector(LakeHuron))
h <- ggplot(huron, aes(year))

h + geom_ribbon(aes(ymin=0, ymax=level))
h + geom_area(aes(y = level))

# Add aesthetic mappings
h +
  geom_ribbon(aes(ymin = level - 1, ymax = level + 1), fill = "grey70") +
  geom_line(aes(y = level))

geom_rug

Rug plots in the margins

Description

A rug plot is a compact visualisation designed to supplement a 2d display with the two 1d marginal distributions. Rug plots display individual cases so are best used with smaller datasets.

Usage

geom_rug(mapping = NULL, data = NULL, stat = "identity",
         position = "identity", ..., sides = "bl", 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.

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 color = "red" or size = 3. They may also be parameters to the paired geom/stat.
sides  A string that controls which sides of the plot the rugs appear on. It can be set to a string containing any of "trbl", for top, right, bottom, and left.

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.

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

Details
The rug lines are drawn with a fixed size (3 are dependent on the overall scale expansion in order not to overplot existing data.

Aesthetics
geom_rug understands the following aesthetics (required aesthetics are in bold):

- alpha
- colour
- group
- linetype
- size
- x
- y

Examples
p <- ggplot(mtcars, aes(wt, mpg)) + geom_point() + geom_rug() + geom_rug(sides="b")  # Rug on bottom only + geom_rug(sides="trbl")  # All four sides

# Use jittering to avoid overplotting for smaller datasets
ggplot(mpg, aes(displ, cty)) + geom_point() + geom_rug() + geom_jitter() + geom_rug(alpha = 1/2, position = "jitter")
Description

gem_segment draws a straight line between points (x, y) and (xend, yend). geom_curve draws a curved line. See the underlying drawing function curveGrob for the parameters that control the curve.

Usage

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

gem_curve(mapping = NULL, data = NULL, stat = "identity", position = "identity", ..., curvature = 0.5, angle = 90, 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.

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 color = "red" or size = 3. They may also be parameters to the paired geom/stat.

arrow specification for arrow heads, as created by arrow().

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

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.

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.

curvature  A numeric value giving the amount of curvature. Negative values produce left-hand curves, positive values produce right-hand curves, and zero produces a straight line.

angle  A numeric value between 0 and 180, giving an amount to skew the control points of the curve. Values less than 90 skew the curve towards the start point and values greater than 90 skew the curve towards the end point.

ncp  The number of control points used to draw the curve. More control points creates a smoother curve.

Details

Both geoms draw a single segment/curve per case. See geom_path if you need to connect points across multiple cases.

Aesthetics

geom_segment understands the following aesthetics (required aesthetics are in bold):

• x
• y
• xend
• yend
• alpha
• colour
• group
• linetype
• size

See Also

geom_path and geom_line for multi-segment lines and paths.

geom_spoke for a segment parameterised by a location (x, y), and an angle and radius.

Examples

b <- ggplot(mtcars, aes(wt, mpg)) +
  geom_point()

df <- data.frame(x1 = 2.62, x2 = 3.57, y1 = 21.0, y2 = 15.0)
b +
  geom_curve(aes(x = x1, y = y1, xend = x2, yend = y2, colour = "curve"), data = df) +
**geom_smooth**

Smoothed conditional means

**Description**

Aids the eye in seeing patterns in the presence of overplotting. `geom_smooth` and `stat_smooth` are effectively aliases: they both use the same arguments. Use `geom_smooth` unless you want to display the results with a non-standard geom.

**Usage**

```r
ggplot(seals, aes(long, lat)) +
  geom_segment(aes(xend = long + delta_long, yend = lat + delta_lat),
               arrow = arrow(length = unit(0.1, "cm"))) +
  borders("state")
```

# You can also use `geom_segment` to recreate plot(type = "h") :

```r
counts <- as.data.frame(table(x = rpois(100, 5)))
counts$x <- as.numeric(as.character(counts$x))
with(counts, plot(x, Freq, type = "h", lwd = 10))
```

```r
ggplot(counts, aes(x, Freq)) +
  geom_segment(aes(xend = x, yend = 0), size = 10, lineend = "butt")
```

**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.

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

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

method
smoothing method (function) to use, eg. "lm", "glm", "gam", "loess", "rlm".
For method = "auto" the smoothing method is chosen based on the size of
the largest group (across all panels). loess is used for more than 1,000 observations;
otherwise gam is used with formula = y ~ s(x, bs = "cs"). Somewhat
anecdotally, loess gives a better appearance, but is O(n^2) in memory, so does
not work for larger datasets.

formula
formula to use in smoothing function, eg. y ~ x, y ~ poly(x, 2), y ~ log(x)

se
display confidence interval around smooth? (TRUE by default, see level to con-
trol

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.

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

geom, stat
Use to override the default connection between geom_smooth and stat_smooth.

n
number of points to evaluate smoother at

span
Controls the amount of smoothing for the default loess smoother. Smaller numbers produce wigglier lines, larger numbers produce smoother lines.

fullrange
should the fit span the full range of the plot, or just the data

level
level of confidence interval to use (0.95 by default)

method.args List of additional arguments passed on to the modelling function defined by

Details
Calculation is performed by the (currently undocumented) predictdf generic and its methods. For
most methods the standard error bounds are computed using the predict method - the exceptions
are loess which uses a t-based approximation, and glm where the normal confidence interval is
constructed on the link scale, and then back-transformed to the response scale.
Aesthetics

geom_smooth understands the following aesthetics (required aesthetics are in bold):

• x
• y
• alpha
• colour
• fill
• group
• linetype
• size
• weight

Computed variables

  y predicted value
  ymin lower pointwise confidence interval around the mean
  ymax upper pointwise confidence interval around the mean
  se standard error

See Also

See individual modelling functions for more details: lm for linear smooths, glm for generalised linear smooths, loess for local smooths

Examples

```r
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_smooth()
```

# Use span to control the "wiggliness" of the default loess smoother
# The span is the fraction of points used to fit each local regression:
```r
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_smooth(span = 0.3)
```

# Instead of a loess smooth, you can use any other modelling function:
```r
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```

```r
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_smooth(method = "lm", formula = y ~ splines::bs(x, 3), se = FALSE)
```
# Smoothes are automatically fit to each group (defined by categorical
# aesthetics or the group aesthetic) and for each facet

```r
ggplot(mpg, aes(displ, hwy, colour = class)) +
  geom_point() +
  geom_smooth(se = FALSE, method = "lm")
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_smooth(span = 0.8) +
  facet_wrap(~drv)
```

```r
binomial_smooth <- function(...) {
  geom_smooth(method = "glm", method.args = list(family = "binomial"), ...)
}
```

# To fit a logistic regression, you need to coerce the values to
# a numeric vector lying between 0 and 1.

```r
ggplot(rpart::kyphosis, aes(Age, Kyphosis)) +
  geom_jitter(height = 0.05) +
  binomial_smooth()
```

```r
ggplot(rpart::kyphosis, aes(Age, as.numeric(Kyphosis) - 1)) +
  geom_jitter(height = 0.05) +
  binomial_smooth()
```

```r
ggplot(rpart::kyphosis, aes(Age, as.numeric(Kyphosis) - 1)) +
  geom_jitter(height = 0.05) +
  binomial_smooth(formula = y ~ splines::ns(x, 2))
```

# But in this case, it's probably better to fit the model yourself
# so you can exercise more control and see whether or not it's a good model

---

**geom_spoke**

*Line segments parameterised by location, direction and distance*

**Description**

This is a polar parameterisation of **geom_segment**. It is useful when you have variables that describe direction and distance.

**Usage**

```r
geom_spoke(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.

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 `color = "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.

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

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

- `x`
- `y`
- `angle`
- `radius`
- `alpha`
- `colour`
- `group`
- `linetype`
- `size`
Examples

```r
df <- expand.grid(x = 1:10, y=1:10)
df$angle <- runif(100, 0, 2*pi)
df$speed <- runif(100, 0, sqrt(0.1 * df$x))

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_spoke(aes(angle = angle), radius = 0.5)

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_spoke(aes(angle = angle, radius = speed))
```

---

**geom_violin**

**Violin plot**

Description

A violin plot is a compact display of a continuous distribution. It is a blend of `geom_boxplot` and `geom_density`: a violin plot is a mirrored density plot displayed in the same way as a boxplot.

Usage

```r
geom_violin(mapping = NULL, data = NULL, stat = "ydensity",
  position = "dodge", ..., draw_quantiles = NULL, trim = TRUE,
  scale = "area", na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)

stat_ydensity(mapping = NULL, data = NULL, geom = "violin",
  position = "dodge", ..., bw = "nrd0", adjust = 1, kernel = "gaussian",
  trim = TRUE, scale = "area", 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.
- **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 color = "red" or size = 3. They may also be parameters to the paired geom/stat.

draw_quantiles If not(NULL) (default), draw horizontal lines at the given quantiles of the density estimate.

trim If TRUE (default), trim the tails of the violins to the range of the data. If FALSE, don’t trim the tails.

scale if "area" (default), all violins have the same area (before trimming the tails). If "count", areas are scaled proportionally to the number of observations. If "width", all violins have the same maximum width.

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.

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

geom, stat Use to override the default connection between geom_violin and stat_ydensity.

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 bw.nrd.

adjust A multiplicate 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.

Aesthetics

geom_violin understands the following aesthetics (required aesthetics are in bold):

• x
• y
• alpha
• colour
• fill
• group
• linetype
• size
• weight
Computed variables

- **density**  
  density estimate
- **scaled**  
  density estimate, scaled to maximum of 1
- **count**  
  density * number of points - probably useless for violin plots
- **violinwidth**  
  density scaled for the violin plot, according to area, counts or to a constant maximum width
- **n**  
  number of points
- **width**  
  width of violin bounding box

References


See Also

- `geom_violin` for examples, and `stat_density` for examples with data along the x axis.

Examples

```r
p <- ggplot(mtcars, aes(factor(cyl), mpg))
p + geom_violin()

p + geom_violin() + geom_jitter(height = 0, width = 0.1)

# Scale maximum width proportional to sample size:
p + geom_violin(scale = "count")

# Scale maximum width to 1 for all violins:
p + geom_violin(scale = "width")

# Default is to trim violins to the range of the data. To disable:
p + geom_violin(trim = FALSE)

# Use a smaller bandwidth for closer density fit (default is 1).
p + geom_violin(adjust = .5)

# Add aesthetic mappings
# Note that violins are automatically dodged when any aesthetic is
# a factor
p + geom_violin(aes(fill = cyl))
p + geom_violin(aes(fill = factor(cyl))))
p + geom_violin(aes(fill = factor(vs))))
p + geom_violin(aes(fill = factor(am))))

# Set aesthetics to fixed value
p + geom_violin(fill = "grey80", colour = ",#3366FF")

# Show quartiles
```


```r
p + geom_violin(draw_quantiles = c(0.25, 0.5, 0.75))

# Scales vs. coordinate transforms -------
if (require("ggplot2movies")) {
  # Scale transformations occur before the density statistics are computed.
  # Coordinate transformations occur afterwards. Observe the effect on the
  # number of outliers.
  m <- ggplot(movies, aes(y = votes, x = rating, group = cut_width(rating, 0.5)))
  m + geom_violin()
  m + geom_violin() + scale_y_log10()
  m + geom_violin() + coord_trans(y = "log10")
  m + geom_violin() + scale_y_log10() + coord_trans(y = "log10")

  # Violin plots with continuous x:
  # Use the group aesthetic to group observations in violins
  ggplot(movies, aes(year, budget)) + geom_violin()
  ggplot(movies, aes(year, budget)) +
    geom_violin(aes(group = cut_width(year, 10)), scale = "width")
}
```

---

**ggplot**

Create a new ggplot

---

**Description**

`ggplot()` initializes a ggplot object. It can be used to declare the input data frame for a graphic and to specify the set of plot aesthetics intended to be common throughout all subsequent layers unless specifically overridden.

**Usage**

```r
ggplot(data = NULL, mapping = aes(), ..., environment = parent.frame())
```

**Arguments**

- `data` Default dataset to use for plot. If not already a data.frame, will be converted to one by `fortify`. If not specified, must be supplied in each layer added to the plot.

- `mapping` Default list of aesthetic mappings to use for plot. If not specified, must be supplied in each layer added to the plot.

- `...` Other arguments passed on to methods. Not currently used.

- `environment` If an variable defined in the aesthetic mapping is not found in the data, `ggplot` will look for it in this environment. It defaults to using the environment in which `ggplot()` is called.
Details

ggplot() is used to construct the initial plot object, and is almost always followed by + to add component to the plot. There are three common ways to invoke ggplot:

1. ggplot(df, aes(x, y, <other aesthetics>))
2. ggplot(df)
3. ggplot()

The first method is recommended if all layers use the same data and the same set of aesthetics, although this method can also be used to add a layer using data from another data frame. See the first example below. The second method specifies the default data frame to use for the plot, but no aesthetics are defined up front. This is useful when one data frame is used predominantly as layers are added, but the aesthetics may vary from one layer to another. The third method initializes a skeleton ggplot object which is fleshed out as layers are added. This method is useful when multiple data frames are used to produce different layers, as is often the case in complex graphics.

Examples

# Generate some sample data, then compute mean and standard deviation
# in each group
df <- data.frame(
  gp = factor(rep(letters[1:3], each = 10)),
  y = rnorm(30)
)
ds <- plyr::ddply(df, "gp", plyr::summarise, mean = mean(y), sd = sd(y))

# The summary data frame ds is used to plot larger red points on top
# of the raw data. Note that we don't need to supply 'data' or 'mapping'
# in each layer because the defaults from ggplot() are used.
ggplot(df, aes(gp, y)) +
  geom_point() +
  geom_point(data = ds, aes(y = mean), colour = 'red', size = 3)

# Same plot as above, declaring only the data frame in ggplot().
# Note how the x and y aesthetics must now be declared in
# each geom_point() layer.
ggplot(df) +
  geom_point(aes(gp, y)) +
  geom_point(data = ds, aes(gp, mean), colour = 'red', size = 3)

# Alternatively we can fully specify the plot in each layer. This
# is not useful here, but can be more clear when working with complex
# multi-dataset graphics
ggplot() +
  geom_point(data = df, aes(gp, y)) +
  geom_point(data = ds, aes(gp, mean), colour = 'red', size = 3) +
  geom_errorbar(
    data = ds,
    aes(gp, mean, ymin = mean - sd, ymax = mean + sd),
    colour = 'red',
    width = 0.4)
Description

Construct a new object with `ggproto`, test with `is.proto`, and access parent methods/fields with `ggproto_parent`.

Usage

```r
ggproto(`_class` = NULL, `_inherit` = NULL, ...)

ggproto_parent(parent, self)

is.ggproto(x)
```

Arguments

- `_class` Class name to assign to the object. This is stored as the class attribute of the object. This is optional: if `NULL` (the default), no class name will be added to the object.
- `_inherit` A `ggproto` object to inherit from. If `NULL`, don’t inherit from any object.
- `...` A list of members in the `ggproto` object.
- `parent, self` Access parent class `parent` of object `self`.
- `x` An object to test.

Details

`ggproto` implements a prototype based OO system which blurs the lines between classes and instances. It is inspired by the `proto` package, but it has some important differences. Notably, it cleanly supports cross-package inheritance, and has faster performance.

In most cases, creating a new OO system to be used by a single package is not a good idea. However, it was the least-bad solution for `ggplot2` because it required the fewest changes to an already complex code base.

Calling methods

`ggproto` methods can take an optional `self` argument: if it is present, it is a regular method; if it’s absent, it’s a "static" method (i.e. it doesn’t use any fields).

Imagine you have a `ggproto` object `Adder`, which has a method `addx = function(self, n) n + self$x`. Then, to call this function, you would use `Adder$addx(10)` – the `self` is passed in automatically by the wrapper function. `self` be located anywhere in the function signature, although customarily it comes first.
Calling methods in a parent

To explicitly call a methods in a parent, use `ggproto_parent(Parent, self).

Examples

```r
Adder <- ggproto("Adder",
x = 0,
add = function(self, n) {
    self$x <- self$x + n
    self$x
}
)
is.ggproto(Adder)

Adder$add(10)
Adder$add(10)

Doubler <- ggproto("Doubler", Adder,
    add = function(self, n) {
        ggproto_parent(Adder, self)$add(n * 2)
    }
)
Doubler$x
Doubler$add(10)
```

ggsave

Save a ggplot (or other grid object) with sensible defaults

Description

`ggsave()` is a convenient function for saving a plot. It defaults to saving the last plot that you displayed, using the size of the current graphics device. It also guesses the type of graphics device from the extension.

Usage

```r
ggsave(filename, plot = last_plot(), device = NULL, path = NULL,
    scale = 1, width = NA, height = NA, units = c("in", "cm", "mm"),
    dpi = 300, limitsize = TRUE, ...)
```

Arguments

- **filename**: File name to create on disk.
- **plot**: Plot to save, defaults to last plot displayed.
- **device**: Device to use. Can be either be a device function (e.g. `png`), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only).
- **path**: Path to save plot to (combined with filename).
scale
width, height, units
Plot size in units ("in", "cm", or "mm"). If not supplied, uses the size of current graphics device.
dpi
Plot resolution. Applies only to raster output types.
limitsize
When TRUE (the default), ggsave will not save images larger than 50x50 inches, to prevent the common error of specifying dimensions in pixels.
...
Other arguments passed on to graphics device.

Examples

```r
## Not run:
ggplot(mtcars, aes(mpg, wt)) + geom_point()

ggsave("mtcars.pdf")
ggsave("mtcars.png")

ggsave("mtcars.pdf", width = 4, height = 4)
ggsave("mtcars.pdf", width = 20, height = 20, units = "cm")

unlink("mtcars.pdf")
unlink("mtcars.png")

# specify device when saving to a file with unknown extension
# (for example a server supplied temporary file)
file <- tempfile()
ggsave(file, device = "pdf")
unlink(file)

## End(Not run)
```

## ggtheme

### Complete themes

**Description**

These are complete themes which control all non-data display. Use `theme` if you just need to tweak the display of an existing theme.

**Usage**

- `theme_grey(base_size = 11, base_family = "")`
- `theme_gray(base_size = 11, base_family = "")`
- `theme_bw(base_size = 11, base_family = "")`
- `theme_linedraw(base_size = 11, base_family = "")`
theme_light(base_size = 11, base_family = "")

theme_dark(base_size = 11, base_family = "")

theme_minimal(base_size = 11, base_family = "")

theme_classic(base_size = 11, base_family = "")

theme_void(base_size = 11, base_family = "")

Arguments

base_size base font size
base_family base font family

Details

tHEME_GRAY The signature ggplot2 theme with a grey background and white gridlines, designed to put the data forward yet make comparisons easy.

theme_bw The classic dark-on-light ggplot2 theme. May work better for presentations displayed with a projector.

theme_linedraw A theme with only black lines of various widths on white backgrounds, reminiscent of a line drawings. Serves a purpose similar to theme_bw. Note that this theme has some very thin lines (« 1 pt) which some journals may refuse.

theme_light A theme similar to theme_linedraw but with light grey lines and axes, to direct more attention towards the data.

theme_dark The dark cousin of theme_light, with similar line sizes but a dark background. Useful to make thin coloured lines pop out.

theme_minimal A minimalistic theme with no background annotations.

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

theme_void A completely empty theme.

Examples

p <- ggplot(mtcars) + geom_point(aes(x = wt, y = mpg,
  colour = factor(gear))) + facet_wrap(~am)
p + theme_gray() # the default
p + theme_bw()
p + theme_linedraw()
p + theme_light()
p + theme_dark()
p + theme_minimal()
p + theme_classic()
p + theme_void()
**guides**

Set guides for each scale

---

**Description**

Guides for each scale can be set scale-by-scale with the `guide` argument, or en masse with `guides()`.

**Usage**

`guides(...)`

**Arguments**

... List of scale name-guide pairs. The guide can either be a string (i.e. "colorbar" or "legend"), or a call to a guide function (i.e. `guide_colourbar` or `guide_legend`) specifying additional arguments.

**Value**

A list containing the mapping between scale and guide.

**See Also**

Other guides: `guide_colourbar`, `guide_legend`

**Examples**

```r
# ggplot object
dat <- data.frame(x = 1:5, y = 1:5, p = 1:5, q = factor(1:5),
                  r = factor(1:5))
p <- ggplot(dat, aes(x, y, colour = p, size = q, shape = r)) + geom_point()

# without guide specification
p

# Show colorbar guide for colour.
# All these examples below have a same effect.
p + guides(colour = "colorbar", size = "legend", shape = "legend")
p + guides(colour = guide_colorbar(), size = guide_legend(),
           shape = guide_legend())
p +
   scale_colour_continuous(guide = "colorbar") +
   scale_size_discrete(guide = "legend") +
   scale_shape(guide = "legend")

# Remove some guides
```
Continuous colour bar guide

description

Colour bar guide shows continuous color scales mapped onto values. Colour bar is available with scale_fill and scale_colour. For more information, see the inspiration for this function: Matlab's colorbar function.

Usage

guide_colourbar(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, ...)

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,
guide_colourbar

draw.ulim = TRUE, draw.llim = TRUE, direction = NULL,
default.unit = "line", reverse = FALSE, order = 0, ...

Arguments

title A character string or expression indicating a title of guide. If NULL, the title is not shown. By default (waiver), the name of the scale object or the name specified in labs is used for the title.
title.position A character string indicating the position of a title. One of "top" (default for a vertical guide), "bottom", "left" (default for a horizontal guide), or "right.”
title.theme A theme object for rendering the title text. Usually the object of element_text is expected. By default, the theme is specified by legend.title in theme or theme.
title.hjust A number specifying horizontal justification of the title text.
title.vjust A number specifying vertical justification of the title text.
label logical. If TRUE then the labels are drawn. If FALSE then the labels are invisible.
label.position A character string indicating the position of a label. One of "top", "bottom" (default for horizontal guide), "left", or "right" (default for vertical guide).
label.theme A theme object for rendering the label text. Usually the object of element_text is expected. By default, the theme is specified by legend.text in theme or theme.
label.hjust A numeric specifying horizontal justification of the label text.
label.vjust A numeric specifying vertical justification of the label text.
barwidth A numeric or a unit object specifying the width of the colorbar. Default value is legend.key.width or legend.key.size in theme or theme.
barheight A numeric or a unit object specifying the height of the colorbar. Default value is legend.key.height or legend.key.size in theme or theme.
nbin A numeric specifying the number of bins for drawing colorbar. A smoother colorbar for a larger value.
raster A logical. If TRUE then the colorbar is rendered as a raster object. If FALSE then the colorbar is rendered as a set of rectangles. Note that not all graphics devices are capable of rendering raster image.
ticks A logical specifying if tick marks on colorbar should be visible.
draw.ulim A logical specifying if the upper limit tick marks should be visible.
draw.llim A logical specifying if the lower limit tick marks should be visible.
direction A character string indicating the direction of the guide. One of "horizontal" or "vertical.”
default.unit A character string indicating unit for barwidth and barheight.
reverse logical. If TRUE the colorbar is reversed. By default, the highest value is on the top and the lowest value is on the bottom
order positive integer less than 99 that specifies the order of this guide among multiple guides. This controls the order in which multiple guides are displayed, not the contents of the guide itself. If 0 (default), the order is determined by a secret algorithm.
...
ignored.
Details

Guides can be specified in each scale_\* or in guides. guide="legend" in scale_\* is syntactic sugar for guide=guide_legend() (e.g. scale_color_manual(guide = "legend")). As for how to specify the guide for each scale in more detail, see guides.

Value

A guide object

See Also

Other guides: guide_legend, guides

Examples

df <- reshape2::melt(outer(1:4, 1:4), varnames = c("X1", "X2"))

p1 <- ggplot(df, aes(X1, X2)) + geom_tile(aes(fill = value))
p2 <- p1 + geom_point(aes(size = value))

# Basic form
p1 + scale_fill_continuous(guide = "colorbar")
p1 + scale_fill_continuous(guide = guide_colorbar())
p1 + guides(fill = guide_colorbar())

# Control styles

# bar size
p1 + guides(fill = guide_colorbar(barwidth = 0.5, barheight = 10))

# no label
p1 + guides(fill = guide_colorbar(label = FALSE))

# no tick marks
p1 + guides(fill = guide_colorbar(ticks = FALSE))

# label position
p1 + guides(fill = guide_colorbar(label.position = "left"))

# label theme
p1 + guides(fill = guide_colorbar(label.theme = element_text(colour = "blue", angle = 0)))

# small number of bins
p1 + guides(fill = guide_colorbar(nbin = 3))

# large number of bins
p1 + guides(fill = guide_colorbar(nbin = 100))

# make top- and bottom-most ticks invisible
p1 + scale_fill_continuous(limits = c(0,20), breaks = c(0, 5, 10, 15, 20),
guide = guide_colorbar(nbin=100, draw.ulim = FALSE, draw.llim = FALSE))
# guides can be controlled independently
p2 +
  scale_fill_continuous(guide = "colorbar") +
  scale_size(guide = "legend")
p2 + guides(fill = "colorbar", size = "legend")

p2 +
  scale_fill_continuous(guide = guide_colorbar(direction = "horizontal")) +
  scale_size(guide = guide_legend(direction = "vertical"))

---

**guide_legend**

**Legend guide**

**Description**

Legend type guide shows key (i.e., geoms) mapped onto values. Legend guides for various scales are integrated if possible.

**Usage**

```r
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,
  ...)```

**Arguments**

- **title**: A character string or expression indicating a title of guide. If NULL, the title is not shown. By default (`waiver`), the name of the scale object or the name specified in `labs` is used for the title.
- **title.position**: A character string indicating the position of a title. One of "top" (default for a vertical guide), "bottom", "left" (default for a horizontal guide), or "right."
- **title.theme**: A theme object for rendering the title text. Usually the object of `element_text` is expected. By default, the theme is specified by `legend.title` in `theme` or `theme`.
- **title.hjust**: A number specifying horizontal justification of the title text.
- **title.vjust**: A number specifying vertical justification of the title text.
- **label**: logical. If TRUE then the labels are drawn. If FALSE then the labels are invisible.
- **label.position**: A character string indicating the position of a label. One of "top", "bottom" (default for horizontal guide), "left", or "right" (default for vertical guide).
- **label.theme**: A theme object for rendering the label text. Usually the object of `element_text` is expected. By default, the theme is specified by `legend.text` in `theme` or `theme`. 
label.hjust  A numeric specifying horizontal justification of the label text.
label.vjust  A numeric specifying vertical justification of the label text.
keywidth  A numeric or a unit object specifying the width of the legend key. Default value is legend.key.width or legend.key.size in theme or theme.
keyheight  A numeric or a unit object specifying the height of the legend key. Default value is legend.key.height or legend.key.size in theme or theme.
direction  A character string indicating the direction of the guide. One of "horizontal" or "vertical."
default.unit  A character string indicating unit for keywidth and keyheight.
override.aes  A list specifying aesthetic parameters of legend key. See details and examples.
nrow  The desired number of rows of legends.
ncol  The desired number of column of legends.
byrow  logical. If FALSE (the default) the legend-matrix is filled by columns, otherwise the legend-matrix is filled by rows.
reverse  logical. If TRUE the order of legends is reversed.
order  positive integer less that 99 that specifies the order of this guide among multiple guides. This controls the order in which multiple guides are displayed, not the contents of the guide itself. If 0 (default), the order is determined by a secret algorithm.

Details

Guides can be specified in each scale_* or in guides. guide="legend" in scale_* is syntactic sugar for guide=guide_legend() (e.g. scale_color_manual(guide = "legend")). As for how to specify the guide for each scale in more detail, see guides.

Value

A guide object

See Also

Other guides: guide_colourbar, guides

Examples

df <- reshape2::melt(outer(1:4, 1:4), varnames = c("X1", "X2"))

p1 <- ggplot(df, aes(X1, X2)) + geom_tile(aes(fill = value))

p2 <- p1 + geom_point(aes(size = value))

# Basic form
p1 + scale_fill_continuous(guide = "legend")
p1 + scale_fill_continuous(guide = guide_legend())
# Guide title
p1 + scale_fill_continuous(guide = guide_legend(title = "V"))  # title text
p1 + scale_fill_continuous(guide = guide_legend(title = NULL))  # no title

# Control styles

# key size
p1 + guides(fill = guide_legend(keywidth = 3, keyheight = 1))

# title position
p1 + guides(fill = guide_legend(title = "LEFT", title.position = "left"))

# title text styles via element_text
p1 + guides(fill =
  guide_legend(
    title.theme = element_text(
      size = 15,
      face = "italic",
      colour = "red",
      angle = 0
    )
  )
)

# label position
p1 + guides(fill = guide_legend(label.position = "left", label.hjust = 1))

# label styles
p1 + scale_fill_continuous(breaks = c(5, 10, 15),
  labels = paste("long", c(5, 10, 15)),
  guide = guide_legend(
    direction = "horizontal",
    title.position = "top",
    label.position = "bottom",
    label.hjust = 0.5,
    label.vjust = 1,
    label.theme = element_text(angle = 90)
  )
)

# Set aesthetic of legend key

# very low alpha value make it difficult to see legend key
p3 <- ggplot(diamonds, aes(carat, price)) +
  geom_point(aes(colour = color), alpha = 1/100)
p3

# override.aes overwrites the alpha
p3 + guides(colour = guide_legend(override.aes = list(alpha = 1)))

# multiple row/col legends
df <- data.frame(x = 1:20, y = 1:20, color = letters[1:20])
A selection of summary functions from \texttt{Hmisc}

\begin{verbatim}
library(ggplot2)
library(Hmisc)

p <- ggplot(df, aes(x, y)) +
  geom_point(aes(colour = color))
p + guides(col = guide_legend(nrow = 8))
p + guides(col = guide_legend(ncol = 8))
p + guides(col = guide_legend(nrow = 8, byrow = TRUE))
p + guides(col = guide_legend(ncol = 8, byrow = TRUE))

# reversed order legend
p + guides(col = guide_legend(reverse = TRUE))
\end{verbatim}

\texttt{hmisc} is a selection of summary functions from \texttt{Hmisc} designed to make them easier to use with \texttt{stat_summary}. See the \texttt{Hmisc} documentation for more details:

- \texttt{smean.cl.boot}
- \texttt{smean.cl.normal}
- \texttt{smean.sdl}
- \texttt{smedian.hilow}

**Usage**

\begin{verbatim}
mean_cl_boot(x, ...)
mean_cl_normal(x, ...)
mean_sdl(x, ...)
median_hilow(x, ...)
\end{verbatim}

**Arguments**

- \texttt{x} a numeric vector
- \texttt{...} other arguments passed on to the respective \texttt{Hmisc} function.

**Value**

A data frame with columns \texttt{y}, \texttt{ymin}, and \texttt{ymax}.
labeller

Examples

```r
x <- rnorm(100)
mean_cl_boot(x)
mean_cl_normal(x)
mean_sdl(x)
median_hilow(x)
```


labeller Construct labelling specification

Description

This function makes it easy to assign different labellers to different factors. The labeller can be a function or it can be a named character vectors that will serve as a lookup table.

Usage

```r
labeller(..., .rows = NULL, .cols = NULL, keep.as.numeric = NULL,
    .multi_line = TRUE, .default = label_value)
```

Arguments

- `...` Named arguments of the form `variable = labeller`. Each labeller is passed to `as_labeller()` and can be a lookup table, a function taking and returning character vectors, or simply a labeller function.
- `.rows, .cols` Labeller for a whole margin (either the rows or the columns). It is passed to `as_labeller()`. When a margin-wide labeller is set, make sure you don’t mention in ... any variable belonging to the margin.
- `keep.as.numeric` Deprecated. All supplied labellers and on-labeller functions should be able to work with character labels.
- `.multi_line` Whether to display the labels of multiple factors on separate lines. This is passed to the labeller function.
- `.default` Default labeller for variables not specified. Also used with lookup tables or non-labeller functions.

Details

In case of functions, if the labeller has class `labeller`, it is directly applied on the data frame of labels. Otherwise, it is applied to the columns of the data frame of labels. The data frame is then processed with the function specified in the `.default` argument. This is intended to be used with functions taking a character vector such as `capitalize`.

Value

A labeller function to supply to `facet_grid` for the argument `labeller`.
See Also

`as_labeller()`, `labellers`

Examples

```r
p1 <- ggplot(mtcars, aes(x = mpg, y = wt)) + geom_point()

# You can assign different labellers to variables:
p1 + facet_grid(vs + am ~ gear,
  labeller = labeller(vs = label_both, am = label_value))

# Or whole margins:
p1 + facet_grid(vs + am ~ gear,
  labeller = labeller(.rows = label_both, .cols = label_value))

# You can supply functions operating on strings:
capitalize <- function(string) {
  substr(string, 1, 1) <- toupper(substr(string, 1, 1))
  string
}
p2 <- ggplot(msleep, aes(x = sleep_total, y = awake)) + geom_point()
p2 + facet_grid(vore ~ conservation, labeller = labeller(vore = capitalize))

# Or use character vectors as lookup tables:
conservation_status <- c(
  cd = "Conservation Dependent",
  en = "Endangered",
  lc = "Least concern",
  nt = "Near Threatened",
  vu = "Vulnerable",
  domesticated = "Domesticated"
)


p2 + facet_grid(vore ~ conservation, labeller = labeller(    .default = capitalize,    conservation = conservation_status))

# In the following example, we rename the levels to the long form,
# then apply a wrap labeller to the columns to prevent cropped text
msleep$conservation2 <- plyr::revalue(msleep$conservation,    conservation_status)

p2 %+% msleep + facet_grid(vore ~ conservation2)
p2 %+% msleep +
  facet_grid(vore ~ conservation2,
    labeller = labeller(conservation2 = label_wrap_gen(10))
  )

# labeller() is especially useful to act as a global labeller. You
labellers

# can set it up once and use it on a range of different plots with
# different facet specifications.

global_labeller <- labeller(
  vore = capitalize,
  conservation = conservation_status,
  conservation2 = label_wrap_gen(10),
  .default = label_both
)

p2 + facet_grid(vore ~ conservation, labeller = global_labeller)
p2 + facet_wrap(~vore, labeller = global_labeller)
p2 %+% msleep + facet_wrap(~conservation2, labeller = global_labeller)

labellers

Useful labeller functions

Description

Labeller functions are in charge of formatting the strip labels of facet grids and wraps. Most of them accept a `multi_line` argument to control whether multiple factors (defined in formulae such as `~first + second`) should be displayed on a single line separated with commas, or each on their own line.

Usage

label_value(labels, multi_line = TRUE)

label_both(labels, multi_line = TRUE, sep = "::")

label_context(labels, multi_line = TRUE, sep = "::")

label_parsed(labels, multi_line = TRUE)

label_wrap_gen(width = 25, multi_line = TRUE)

Arguments

labels Data frame of labels. Usually contains only one element, but facetting over multiple factors entails multiple label variables.

multi_line Whether to display the labels of multiple factors on separate lines.

sep String separating variables and values.

width Maximum number of characters before wrapping the strip.
Details

`label_value()` only displays the value of a factor while `label_both()` displays both the variable name and the factor value. `label_context()` is context-dependent and uses `label_value()` for single factor facetting and `label_both()` when multiple factors are involved. `label_wrap_gen()` uses `strwrap()` for line wrapping.

`label_parsed()` interprets the labels as plotmath expressions. `label_bquote()` offers a more flexible way of constructing plotmath expressions. See examples and `bquote()` for details on the syntax of the argument.

Writing New Labeller Functions

Note that an easy way to write a labeller function is to transform a function operating on character vectors with `as_labeller()`.

A labeller function accepts a data frame of labels (character vectors) containing one column for each factor. Multiple factors occur with formula of the type `first + second`.

The return value must be a rectangular list where each `row` characterises a single facet. The list elements can be either character vectors or lists of plotmath expressions. When multiple elements are returned, they get displayed on their own new lines (i.e., each facet gets a multi-line strip of labels).

To illustrate, let's say your labeller returns a list of two character vectors of length 3. This is a rectangular list because all elements have the same length. The first facet will get the first elements of each vector and display each of them on their own line. Then the second facet gets the second elements of each vector, and so on.

If it's useful to your labeller, you can retrieve the type attribute of the incoming data frame of labels. The value of this attribute reflects the kind of strips your labeller is dealing with: "cols" for columns and "rows" for rows. Note that `facet_wrap()` has columns by default and rows when the strips are switched with the `switch` option. The facet attribute also provides metadata on the labels. It takes the values "grid" or "wrap".

For compatibility with `labeller()`, each labeller function must have the `labeller` S3 class.

See Also

`labeller()`, `as_labeller()`, `label_bquote()`

Examples

```r
mtcars$cyl2 <- factor(mtcars$cyl, labels = c("alpha", "beta", "gamma"))
p <- ggplot(mtcars, aes(wt, mpg)) + geom_point()

# Displaying only the values
p + facet_grid(. ~ cyl)
p + facet_grid(. ~ cyl, labeller = label_value)

# Displaying both the values and the variables
p + facet_grid(. ~ cyl, labeller = label_both)

# Displaying only the values or both the values and variables
```
`label_bquote`

# depending on whether multiple factors are facetted over
p + facet_grid(am ~ vs + cyl, labeller = label_context)

# Interpreting the labels as plotmath expressions
p + facet_grid(., ~ cyl)
p + facet_grid(., ~ cyl, labeller = label_parsed)
p + facet_wrap(~ vs + cyl, labeller = label_parsed)

---

**label_bquote**  
*Label with mathematical expressions*

**Description**

`label_bquote()` offers a flexible way of labelling facet rows or columns with plotmath expressions. Backquoted variables will be replaced with their value in the facet.

**Usage**

`label_bquote(rows = NULL, cols = NULL, default = label_value)`

**Arguments**

- `rows`  
  Backquoted labelling expression for rows.

- `cols`  
  Backquoted labelling expression for columns.

- `default`  
  Default labeller function for the rows or the columns when no plotmath expression is provided.

**See Also**

`labellers`, `labeller()`.

**Examples**

# The variables mentioned in the plotmath expression must be
# backquoted and referred to by their names.
p <- ggplot(mtcars, aes(wt, mpg)) + geom_point()
p + facet_grid(vs ~ ., labeller = label_bquote(alpha ~ .(vs)))
p + facet_grid(., ~ vs, labeller = label_bquote(cols = .(vs) ^ .(vs)))
p + facet_grid(., ~ vs + am, labeller = label_bquote(cols = .(am) ^ .(vs)))
Modify axis, legend, and plot labels

Description

Good labels are critical for making your plots accessible to a wider audience. Ensure the axis and legend labels display the full variable name. Use the plot title and subtitle to explain the main findings. It’s common to use the caption to provide information about the data source.

Usage

```r
labs(...)  
  xlab(label)  
  ylab(label)  
  ggtitle(label, subtitle = NULL)
```

Arguments

- `...`: A list of new name-value pairs. The name should either be an aesthetic, or one of "title", "subtitle", or "caption".
- `label`: The text for the axis, plot title or caption below the plot.
- `subtitle`: the text for the subtitle for the plot which will be displayed below the title. Leave NULL for no subtitle.

Details

You can also set axis and legend labels in the individual scales (using the first argument, the name). I recommend doing that if you’re changing other scale options.

Examples

```r
p <- ggplot(mtcars, aes(mpg, wt, colour = cyl)) + geom_point()  
p + labs(colour = "Cylinders")  
p + labs(x = "New x label")
```

# The plot title appears at the top-left, with the subtitle
# display in smaller text underneath it
```

```
```r
p + labs(title = "New plot title")  
p + labs(title = "New plot title", subtitle = "A subtitle")
```

# The caption appears in the bottom-right, and is often used for
# sources, notes or copyright
```
```
```r
p + labs(caption = "(based on data from ...)")
```
**Description**

This is a shortcut for supplying the `limits` argument to the individual scales. Note that, by default, any values outside the limits will be replaced with `NA`.

**Usage**

```r
lims(...)  
xlim(...)  
ylim(...)
```

**Arguments**

`...`  
A name-value pair. The name must be an aesthetic, and the value must be either a length-2 numeric, a character, a factor, or a date/time.  
A numeric value will create a continuous scale. If the larger value comes first, the scale will be reversed. You can leave one value as `NA` to compute from the range of the data.  
A character or factor value will create a discrete scale.  
A date-time value will create a continuous date/time scale.

**See Also**

For changing x or y axis limits without dropping data observations, see `coord_cartesian`. To expand the range of a plot to always include certain values, see `expand_limits`.

**Examples**

```r
# Zoom into a specified area  
ggplot(mtcars, aes(mpg, wt)) +  
  geom_point() +  
  xlim(15, 20)

# reverse scale  
ggplot(mtcars, aes(mpg, wt)) +  
  geom_point() +  
  xlim(20, 15)

# with automatic lower limit  
ggplot(mtcars, aes(mpg, wt)) +  
  geom_point() +  
  xlim(NA, 20)
```
# You can also supply limits that are larger than the data.
# This is useful if you want to match scales across different plots
small <- subset(mtcars, cyl == 4)
big <- subset(mtcars, cyl > 4)

ggplot(small, aes(mpg, wt, colour = factor(cyl))) +
  geom_point() +
  lims(colour = c("4", "6", "8"))

 ggplot(big, aes(mpg, wt, colour = factor(cyl))) +
  geom_point() +
  lims(colour = c("4", "6", "8"))

---

luv_colours colors() in Luv space

Description
All built-in colors() translated into Luv colour space.

Usage
luv_colours

Format
A data frame with 657 observations and 4 variables:

L, u, v  Position in Luv colour space
col  Colour name

---

margin  Theme elements

Description
In conjunction with the theme system, the element_ functions specify the display of how non-data components of the plot are drawn.

- element_blank: draws nothing, and assigns no space.
- element_rect: borders and backgrounds.
- element_line: lines.
- element_text: text.

rel() is used to specify sizes relative to the parent, margins() is used to specify the margins of elements.
Usage

```
margin(t = 0, r = 0, b = 0, l = 0, unit = "pt")
```

element_blank()

element_rect(fill = NULL, colour = NULL, size = NULL, linetype = NULL, color = NULL, inherit.blank = FALSE)

element_line(colour = NULL, size = NULL, linetype = NULL, lineend = NULL, color = NULL, arrow = NULL, inherit.blank = FALSE)

element_text(family = NULL, face = NULL, colour = NULL, size = NULL, hjust = NULL, vjust = NULL, angle = NULL, lineheight = NULL, color = NULL, margin = NULL, debug = NULL, inherit.blank = FALSE)

rel(x)

Arguments

t, r, b, l Dimensions of each margin. (To remember order, think trouble).

unit Default units of dimensions. Defaults to "pt" so it can be most easily scaled with the text.

fill Fill colour.

colour, color Line/border colour. Color is an alias for colour.

size Line/border size in mm; text size in pts.

linetype Line type. An integer (0:8), a name (blank, solid, dashed, dotted, dotdash, long-dash, twodash), or a string with an even number (up to eight) of hexadecimal digits which give the lengths in consecutive positions in the string.

inherit.blank Should this element inherit the existence of an element_blank among its parents? If TRUE the existence of a blank element among its parents will cause this element to be blank as well. If FALSE any blank parent element will be ignored when calculating final element state.

lineend Line end style (round, butt, square)

arrow Arrow specification, as created by arrow

family Font family

face Font face ("plain", "italic", "bold", "bold.italic")

hjust Horizontal justification (in [0, 1])

vjust Vertical justification (in [0, 1])

angle Angle (in [0, 360])

lineheight Line height

margin Margins around the text. See margin for more details. When creating a theme, the margins should be placed on the side of the text facing towards the center of the plot.
mean_se

Calculate mean and standard error

Description
For use with stat_summary

Usage
mean_se(x, mult = 1)

Arguments
x numeric vector
mult number of multiples of standard error

debug If TRUE, aids visual debugging by drawing a solid rectangle behind the complete text area, and a point where each label is anchored.
x A single number specifying size relative to parent element.

Value
An S3 object of class element, rel, or margin.

Examples
plot <- ggplot(mpg, aes(displ, hwy)) + geom_point()

plot + theme(
  panel.background = element_blank(),
  axis.text = element_blank()
)

plot + theme(
  axis.text = element_text(colour = "red", size = rel(1.5))
)

plot + theme(
  axis.line = element_line(arrow = arrow())
)

plot + theme(
  panel.background = element_rect(fill = "white"),
  plot.margin = margin(2, 2, 2, 2, "cm"),
  plot.background = element_rect(
    fill = "grey90",
    colour = "black",
    size = 1
  )
)
Value

A data frame with columns `y`, `ymin`, and `ymax`.

Examples

```r
x <- rnorm(100)
mean_se(x)
```

midwest  Midwest demographics

Description

Demographic information of midwest counties

Usage

`midwest`

Format

A data frame with 437 rows and 28 variables

- **PID**
- **county**
- **state**
- **area**
- **poptotal** Total population
- **popdensity** Population density
- **popwhite** Number of whites.
- **popblack** Number of blacks.
- **popamerindian** Number of American Indians.
- **popasian** Number of Asians.
- **popother** Number of other races.
- **percwhite** Percent white.
- **percblack** Percent black.
- **percamerindan** Percent American Indian.
- **percasian** Percent Asian.
- **percother** Percent other races.
- **popadults** Number of adults.
- **perchsd**
- **percollege** Percent college educated.
mpg

percprof  Percent profession.
poppovertyknown
percpovertyknown
percbelowpoverty
percchildbelowpovert
percadultpoverty
percelderlypoverty
inmetro  In a metro area.
category

mpg

*Fuel economy data from 1999 and 2008 for 38 popular models of car*

**Description**

This dataset contains a subset of the fuel economy data that the EPA makes available on [http://fueleconomy.gov](http://fueleconomy.gov). It contains only models which had a new release every year between 1999 and 2008 - this was used as a proxy for the popularity of the car.

**Usage**

mpg

**Format**

A data frame with 234 rows and 11 variables

manufacturer
model  model name
displ  engine displacement, in litres
year  year of manufacture
cyl  number of cylinders
trans  type of transmission
drv  f = front-wheel drive, r = rear wheel drive, 4 = 4wd
cty  city miles per gallon
hwy  highway miles per gallon
fl  fuel type
class  "type" of car
msleep

An updated and expanded version of the mammals sleep dataset

Description

This is an updated and expanded version of the mammals sleep dataset. Updated sleep times and weights were taken from V. M. Savage and G. B. West. A quantitative, theoretical framework for understanding mammalian sleep. Proceedings of the National Academy of Sciences, 104 (3):1051-1056, 2007.

Usage

msleep

Format

A data frame with 83 rows and 11 variables

name common name
genus
vore carnivore, omnivore or herbivore?
order
conservation the conservation status of the animal
sleep_total total amount of sleep, in hours
sleep_rem rem sleep, in hours
sleep_cycle length of sleep cycle, in hours
awake amount of time spent awake, in hours
brainwt brain weight in kilograms
bodywt body weight in kilograms

Details

Additional variables order, conservation status and vore were added from wikipedia.
position_dodge

Dodge overlapping objects side-to-side

Description

Dodging preserves the vertical position of an geom while adjusting the horizontal position.

Usage

position_dodge(width = NULL)

Arguments

width

Dodging width, when different to the width of the individual elements. This is useful when you want to align narrow geoms with wider geoms. See the examples.

See Also

Other position adjustments: position_identity, position_jitterdodge, position_jitter, position_nudge, position_stack

Examples

ggplot(mtcars, aes(factor(cyl), fill = factor(vs))) +
  geom_bar(position = "dodge")

ggplot(diamonds, aes(price, fill = cut)) +
  geom_histogram(position = "dodge")
  # see ?geom_boxplot and ?geom_bar for more examples

  # In this case a frequency polygon is probably a better choice
  ggplot(diamonds, aes(price, colour = cut)) +
  geom_freqpoly()

  # Dodging with various widths -------------------------------------
  # To dodge items with different widths, you need to be explicit
  df <- data.frame(x = c("a","a","b","b"), y = 2:5, g = rep(1:2, 2))
  p <- ggplot(df, aes(x, y, group = g)) +
  geom_col(position = "dodge", fill = "grey50", colour = "black")
  p

  # A line range has no width:
  p + geom_linerange(aes(ymin = y - 1, ymax = y + 1), position = "dodge")

  # So you must explicitly specify the width
  p + geom_linerange(
    aes(ymin = y - 1, ymax = y + 1),
position_identity

Don’t adjust position

Description

Don’t adjust position

Usage

position_identity()

See Also

Other position adjustments: `position_dodge, position_jitterdodge, position_jitter, position_nudge, position_stack`

position_jitter

Jitter points to avoid overplotting

Description

Counterintuitively adding random noise to a plot can sometimes make it easier to read. Jittering is particularly useful for small datasets with at least one discrete position.

Usage

position_jitter(width = NULL, height = NULL)
Arguments

- `width`, `height`  
  Amount of vertical and horizontal jitter. The jitter is added in both positive and negative directions, so the total spread is twice the value specified here.
  
  If omitted, defaults to 40% of the resolution of the data: this means the jitter values will occupy 80% of the implied bins. Categorical data is aligned on the integers, so a width or height of 0.5 will spread the data so it’s not possible to see the distinction between the categories.

See Also

Other position adjustments: `position_dodge`, `position_identity`, `position_jitterdodge`, `position_nudge`, `position_stack`

Examples

```r
# Jittering is useful when you have a discrete position, and a relatively small number of points
# take up as much space as a boxplot or a bar
ggplot(mpg, aes(class, hwy)) +
  geom_boxplot(colour = "grey50") +
  geom_jitter()

# If the default jittering is too much, as in this plot:
ggplot(mtcars, aes(am, vs)) +
  geom_jitter()

# You can adjust it in two ways
ggplot(mtcars, aes(am, vs)) +
  geom_jitter(width = 0.1, height = 0.1)
  geom_jitter(position = position_jitter(width = 0.1, height = 0.1))
```

Description

This is primarily used for aligning points generated through `geom_point()` with dodged boxplots (e.g., a `geom_boxplot()` with a fill aesthetic supplied).

Usage

```r
position_jitterdodge(jitter.width = NULL, jitter.height = 0,
                      dodge.width = 0.75)
```
position_nudge

Arguments

jitter.width  degree of jitter in x direction. Defaults to 40% of the resolution of the data.
jitter.height  degree of jitter in y direction. Defaults to 0.
dodge.width    the amount to dodge in the x direction. Defaults to 0.75, the default position_dodge() width.

See Also

Other position adjustments: position_dodge, position_identity, position_jitter, position_nudge, position_stack

Examples

dsub <- diamonds[ sample(nrow(diamonds), 1000), ]
ggplot(dsub, aes(x = cut, y = carat, fill = clarity)) +
  geom_boxplot(outlier.size = 0) +
  geom_point(pch = 21, position = position_jitterdodge())

position_nudge  Nudge points a fixed distance

Description

position_nudge is generally useful for adjusting the position of items on discrete scales by a small amount. Nudging is built in to geom_text because it’s so useful for moving labels a small distance from what they’re labelling.

Usage

position_nudge(x = 0, y = 0)

Arguments

x, y  Amount of vertical and horizontal distance to move.

See Also

Other position adjustments: position_dodge, position_identity, position_jitterdodge, position_jitter, position_stack
Examples

df <- data.frame(
  x = c(1,3,2,5),
  y = c("a","c","d","c")
)

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text(aes(label = y))

ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text(aes(label = y), position = position_nudge(y = -0.1))

# Or, in brief

# position_stack

position_stack

Stack overlapping objects on top of each another

Description

position_stack() stacks bars on top of each other; position_fill() stacks bars and standardises each stack to have constant height.

Usage

position_stack(vjust = 1, reverse = FALSE)

position_fill(vjust = 1, reverse = FALSE)

Arguments

vjust Vertical adjustment for geoms that have a position (like points or lines), not a dimension (like bars or areas). Set to 0 to align with the bottom, 0.5 for the middle, and 1 (the default) for the top.

reverse If TRUE, will reverse the default stacking order. This is useful if you’re rotating both the plot and legend.

Details

position_fill() and position_stack() automatically stack values in reverse order of the group aesthetic, which for bar charts is usually defined by the fill aesthetic (the default group aesthetic is formed by the combination of all discrete aesthetics except for x and y). This default ensures that bar colours align with the default legend.

There are three ways to override the defaults depending on what you want:
1. Change the order of the levels in the underlying factor. This will change the stacking order, and the order of keys in the legend.

2. Set the legend breaks to change the order of the keys without affecting the stacking.

3. Manually set the group aesthetic to change the stacking order without affecting the legend.

Stacking of positive and negative values are performed separately so that positive values stack upwards from the x-axis and negative values stack downward.

See Also

See `geom_bar` and `geom_area` for more examples.

Other position adjustments: `position_dodge`, `position_identity`, `position_jitterdodge`, `position_jitter`, `position_nudge`

Examples

```
# Stacking and filling

# Stacking is the default behaviour for most area plots.
# Fill makes it easier to compare proportions
ggplot(mtcars, aes(factor(cyl), fill = factor(vs))) +
    geom_bar()

# You can change the order of the levels in the legend using the scale
ggplot(series, aes(time, value)) +
    geom_area(aes(fill = type)) +
    scale_fill_discrete(breaks = c('a', 'b', 'c', 'd'))
```
# Non-area plots ---------------------------------------------------------------

# When stacking across multiple layers it's a good idea to always set
# the `group` aesthetic in the ggplot() call. This ensures that all layers
# are stacked in the same way.

```r
ggplot(series, aes(time, value, group = type)) +
  geom_line(aes(colour = type), position = "stack") +
  geom_point(aes(colour = type), position = "stack")
```

```r
ggplot(series, aes(time, value, group = type)) +
  geom_area(aes(fill = type)) +
  geom_line(aes(group = type), position = "stack")
```

# You can also stack labels, but the default position is suboptimal.

```r
ggplot(series, aes(time, value, group = type)) +
  geom_area(aes(fill = type)) +
  geom_text(aes(label = type), position = "stack")
```

# You can override this with the vjust parameter. A vjust of 0.5
# will center the labels inside the corresponding area

```r
ggplot(series, aes(time, value, group = type)) +
  geom_area(aes(fill = type)) +
  geom_text(aes(label = type), position = position_stack(vjust = 0.5))
```

# Negative values -------------------------------------------------------------

def <- tibble::tribble(
  ~x, ~y, ~grp,
  "a", 1, "x",
  "a", 2, "y",
  "b", 1, "x",
  "b", 3, "y",
  "b", -1, "y"
)

```r
ggplot(data = def, aes(x, y, group = grp)) +
  geom_col(aes(fill = grp), position = position_stack(reverse = TRUE)) +
  geom_hline(yintercept = 0)
```

```r
ggplot(data = def, aes(x, y, group = grp)) +
  geom_col(aes(fill = grp)) +
  geom_hline(yintercept = 0) +
  geom_text(aes(label = grp), position = position_stack(vjust = 0.5))
```

---

**presidential**

**Terms of 11 presidents from Eisenhower to Obama**

**Description**

The names of each president, the start and end date of their term, and their party of 11 US presidents
from Eisenhower to Obama.
print.ggplot

Usage

presidential

Format

A data frame with 11 rows and 4 variables

Description

Generally, you do not need to print or plot a ggplot2 plot explicitly: the default top-level print method will do it for you. You will, however, need to call print() explicitly if you want to draw a plot inside a function or for loop.

Usage

## S3 method for class 'ggplot'
print(x, newpage = is.null(vp), vp = NULL, ...)

## S3 method for class 'ggplot'
plot(x, newpage = is.null(vp), vp = NULL, ...)

Arguments

x plot to display
newpage draw new (empty) page first?
vp viewport to draw plot in
... other arguments not used by this method

Value

Invisibly returns the result of ggplot_build, which is a list with components that contain the plot itself, the data, information about the scales, panels etc.

Examples

colours <- list(~class, ~drv, ~fl)

# Doesn't seem to do anything!
for (colour in colours) {
  ggplot(mpg, aes_(~ displ, ~ hwy, colour = colour)) +
  geom_point()
}

# Works when we explicitly print the plots
for (colour in colours) {
    print(ggplot(mpg, aes_(~ displ, ~ hwy, colour = colour)) +
          geom_point())
}

---

**print.ggproto**  
*Format or print a ggproto object*

**Description**

If a ggproto object has a `print` method, this will call that method. Otherwise, it will print out the members of the object, and optionally, the members of the inherited objects.

**Usage**

```r
## S3 method for class 'ggproto'
print(x, ..., flat = TRUE)

## S3 method for class 'ggproto'
format(x, ..., flat = TRUE)
```

**Arguments**

- `x`  
  A ggproto object to print.

- `...`  
  If the ggproto object has a `print` method, further arguments will be passed to it. Otherwise, these arguments are unused.

- `flat`  
  If TRUE (the default), show a flattened list of all local and inherited members. If FALSE, show the inheritance hierarchy.

**Examples**

```r
Dog <- ggproto(
    print = function(self, n) {
        cat("Woof!\n")
    }\n)

Dog

cat(format(Dog), "\n")
```
**qplot**

*Quick plot*

---

**Description**

qplot is a shortcut designed to be familiar if you’re used to base plot(). It’s a convenient wrapper for creating a number of different types of plots using a consistent calling scheme. It’s great for allowing you to produce plots quickly, but I highly recommend learning ggplot() as it makes it easier to create complex graphics.

**Usage**

```r
qplot(x, y = NULL, ..., data, facets = NULL, margins = FALSE,
      geom = "auto", xlim = c(NA, NA), ylim = c(NA, NA), log = "",
      main = NULL, xlab = deparse(substitute(x)),
      ylab = deparse(substitute(y)), asp = NA, stat = NULL, position = NULL)

quickplot(x, y = NULL, ..., data, facets = NULL, margins = FALSE,
          geom = "auto", xlab = c(NA, NA), ylim = c(NA, NA), log = "",
          main = NULL, xlab = deparse(substitute(x)),
          ylab = deparse(substitute(y)), asp = NA, stat = NULL, position = NULL)
```

**Arguments**

- **x, y, ...** Aesthetics passed into each layer
- **data** Data frame to use (optional). If not specified, will create one, extracting vectors from the current environment.
- **facets** Faceting formula to use. Picks `facet_wrap` or `facet_grid` depending on whether the formula is one- or two-sided
- **margins** See `facet_grid`: display marginal facets?
- **geom** Character vector specifying geom(s) to draw. Defaults to "point" if x and y are specified, and "histogram" if only x is specified.
- **xlim, ylim** X and y axis limits
- **log** Which variables to log transform ("x", "y", or "xy")
- **main, xlab, ylab** Character vector (or expression) giving plot title, x axis label, and y axis label respectively.
- **asp** The y/x aspect ratio
- **stat, position** DEPRECATED.
Examples

# Use data from data.frame
qplot(mpg, wt, data = mtcars)
qplot(mpg, wt, data = mtcars, colour = cyl)
qplot(mpg, wt, data = mtcars, size = cyl)
qplot(mpg, wt, data = mtcars, facets = vs ~ am)

qplot(1:10, rnorm(10), colour = runif(10))
qplot(1:10, letters[1:10])
mod <- lm(mpg ~ wt, data = mtcars)
qplot(resid(mod), fitted(mod))

f <- function() {
  a <- 1:10
  b <- a ^ 2
  qplot(a, b)
}
f()

# To set aesthetics, wrap in I()
qplot(mpg, wt, data = mtcars, colour = I("red"))

# qplot will attempt to guess what geom you want depending on the input
# both x and y supplied = scatterplot
qplot(mpg, wt, data = mtcars)
# just x supplied = histogram
qplot(mpg, data = mtcars)
# just y supplied = scatterplot, with x = seq_along(y)
qplot(y = mpg, data = mtcars)

# Use different geoms
qplot(mpg, wt, data = mtcars, geom = "path")
qplot(factor(cyl), wt, data = mtcars, geom = c("boxplot", "jitter"))
qplot(mpg, data = mtcars, geom = "dotplot")

resolution(x, zero = TRUE)

resolution

Compute the "resolution" of a numeric vector

Description

The resolution is the smallest non-zero distance between adjacent values. If there is only one unique value, then the resolution is defined to be one. If x is an integer vector, then it is assumed to represent a discrete variable, and the resolution is 1.

Usage

resolution(x, zero = TRUE)
scale_alpha

Arguments

- **x** numeric vector
- **zero** should a zero value be automatically included in the computation of resolution

Examples

- `resolution(1:10)`
- `resolution((1:10) - 0.5)`
- `resolution((1:10) - 0.5, FALSE)`

# Note the difference between numeric and integer vectors
- `resolution(c(2, 10, 20, 50))`
- `resolution(c(2L, 10L, 20L, 50L))`

Description

Alpha-transparency scales are not tremendously useful, but can be a convenient way to visually down-weight less important observations. `scale_alpha` is an alias for `scale_alpha_continuous` since that is the most common use of alpha, and it saves a bit of typing.

Usage

- `scale_alpha(..., range = c(0.1, 1))`
- `scale_alpha_continuous(..., range = c(0.1, 1))`
- `scale_alpha_discrete(..., range = c(0.1, 1))`

Arguments

- **...** Other arguments passed on to `continuous_scale` or `discrete_scale` as appropriate, to control name, limits, breaks, labels and so forth.
- **range** Output range of alpha values. Must lie between 0 and 1.

See Also

Other colour scales: `scale_colour_brewer`, `scale_colour_gradient`, `scale_colour_grey`, `scale_colour_hue`

Examples

- `p <- ggplot(mpg, aes(displ, hwy)) + geom_point(aes(alpha = year))`
- `p + scale_alpha("cylinders")`
- `p + scale_alpha(range = c(0.4, 0.8))`
scale_colour_brewer

Sequential, diverging and qualitative colour scales from colorbrewer.org

Description

The brewer scales provides sequential, diverging and qualitative colour schemes from ColorBrewer. These are particularly well suited to display discrete values on a map. See http://colorbrewer2.org for more information.

Usage

scale_colour_brewer(..., type = "seq", palette = 1, direction = 1)

scale_fill_brewer(..., type = "seq", palette = 1, direction = 1)

scale_colour_distiller(..., type = "seq", palette = 1, direction = -1,
values = NULL, space = "Lab", na.value = "grey50",
guide = "colourbar")

dl_scale_fill_distiller(..., type = "seq", palette = 1, direction = -1,
values = NULL, space = "Lab", na.value = "grey50",
guide = "colourbar")

Arguments

... Other arguments passed on to discrete_scale to control name, limits, breaks, labels and so forth.
type One of seq (sequential), div (diverging) or qual (qualitative)
palette If a string, will use that named palette. If a number, will index into the list of palettes of appropriate type
direction Sets the order of colors in the scale. If 1, the default, colors are as output by brewer.pal. If -1, the order of colors is reversed.
values if colours should not be evenly positioned along the gradient this vector gives the position (between 0 and 1) for each colour in the colours vector. See rescale for a convience function to map an arbitrary range to between 0 and 1.
space colour space in which to calculate gradient. Must be "Lab" - other values are deprecated.
na.value Colour to use for missing values
guide Type of legend. Use "colourbar" for continuous colour bar, or "legend" for discrete colour legend.

Details

The brewer scales were carefully designed and tested on discrete data. They were not designed to be extended to continuous data, but results often look good. Your mileage may vary.
scale_colour_brewer

Palettes

The following palettes are available for use with these scales:

**Diverging**  BrBG, PiYG, PRGn, PuOr, RdBu, RdGy, RdYlBu, RdYlGn, Spectral

**Qualitative**  Accent, Dark2, Paired, Pastel1, Pastel2, Set1, Set2, Set3

**Sequential**  Blues, BuGn, BuPu, GnBu, Greens, Greys, Oranges, OrRd, PuBu, PuBuGn, PuRd, Purples, RdPu, Reds, YlGn, YlGnBu, YiOrBr, YiOrRd

**Note**

The distiller scales extends brewer to continuous scales by smoothly interpolate 6 colours from any palette to a continuous scale.

**See Also**

Other colour scales: `scale_alpha, scale_colour_gradient, scale_colour_grey, scale_colour_hue`

**Examples**

dsamp <- diamonds[sample(nrow(diamonds), 1000), ]

(d <- ggplot(dsamp, aes(carat, price)) +
    geom_point(aes(colour = clarity)))

d + scale_colour_brewer()

# Change scale label

d + scale_colour_brewer("Diamond\nclarity")

# Select brewer palette to use, see ?scales::brewer_pal for more details

d + scale_colour_brewer(palette = "Greens")

d + scale_colour_brewer(palette = "Set1")

# scale_fill_brewer works just the same as # scale_colour_brewer but for fill colours

p <- ggplot(diamonds, aes(x = price, fill = cut)) +
    geom_histogram(position = "dodge", binwidth = 1000)

p + scale_fill_brewer()

# the order of colour can be reversed

p + scale_fill_brewer(direction = -1)

# the brewer scales look better on a darker background

p + scale_fill_brewer(direction = -1) + theme_dark()

# Use distiller variant with continous data

v <- ggplot(faithfuld) +
    geom_tile(aes(waiting, eruptions, fill = density))

v

v + scale_fill_distiller()

v + scale_fill_distiller(palette = "Spectral")
scale_colour_gradient  Gradient colour scales

Description

scale_*_gradient creates a two colour gradient (low-high), scale_*_gradient2 creates a diverging colour gradient (low-mid-high), scale_*_gradientn creates a n-colour gradient.

Usage

scale_colour_gradient(..., low = "#132B43", high = "#56B1F7", space = "Lab", na.value = "grey50", guide = "colourbar")

scale_fill_gradient(..., low = "#132B43", high = "#56B1F7", space = "Lab", na.value = "grey50", guide = "colourbar")

scale_colour_gradient2(..., low = muted("red"), mid = "white", high = muted("blue"), midpoint = 0, space = "Lab", na.value = "grey50", guide = "colourbar")

scale_fill_gradient2(..., low = muted("red"), mid = "white", high = muted("blue"), midpoint = 0, space = "Lab", na.value = "grey50", guide = "colourbar")

scale_colour_gradientn(..., colours, values = NULL, space = "Lab", na.value = "grey50", guide = "colourbar", colors)

scale_fill_gradientn(..., colours, values = NULL, space = "Lab", na.value = "grey50", guide = "colourbar", colors)

Arguments

... Other arguments passed on to continuous_scale to control name, limits, breaks, labels and so forth.
low, high Colours for low and high ends of the gradient.
space colour space in which to calculate gradient. Must be "Lab" - other values are deprecated.
na.value Colour to use for missing values
guide Type of legend. Use "colourbar" for continuous colour bar, or "legend" for discrete colour legend.
mid colour for mid point
midpoint The midpoint (in data value) of the diverging scale. Defaults to 0.
colours, colors Vector of colours to use for n-colour gradient.
values if colours should not be evenly positioned along the gradient this vector gives the position (between 0 and 1) for each colour in the colours vector. See rescale for a convience function to map an arbitrary range to between 0 and 1.

Details

Default colours are generated with munsell and mns1(c("2.5PB 2/4", "2.5PB 7/10"). Generally, for continuous colour scales you want to keep hue constant, but vary chroma and luminance. The munsell package makes this easy to do using the Munsell colour system.

See Also

seq_gradient_pal for details on underlying palette
Other colour scales: scale_alpha, scale_colour_brewer, scale_colour_grey, scale_colour_hue

Examples

df <- data.frame(  x = runif(100),  y = runif(100),  z1 = rnorm(100),  z2 = abs(rnorm(100)) )

# Default colour scale colours from light blue to dark blue  ggplot(df, aes(x, y)) +    geom_point(aes(colour = z2))

# For diverging colour scales use gradient2  ggplot(df, aes(x, y)) +    geom_point(aes(colour = z1)) +    scale_colour_gradient2()

# Use your own colour scale with gradientn  ggplot(df, aes(x, y)) +    geom_point(aes(colour = z1)) +    scale_colour_gradientn(colours = terrain.colors(10))

# Equivalent fill scales do the same job for the fill aesthetic  ggplot(faithful, aes(waiting, eruptions)) +    geom_raster(aes(fill = density)) +    scale_fill_gradientn(colours = terrain.colors(10))

# Adjust colour choices with low and high  ggplot(df, aes(x, y)) +    geom_point(aes(colour = z2)) +    scale_colour_gradient(low = "white", high = "black")
# Avoid red-green colour contrasts because ~10% of men have difficulty # seeing them
scale_colour_grey

Description

Based on `gray.colors`. This is black and white equivalent of `scale_colour_gradient`.

Usage

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

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

Arguments

- `...`: Other arguments passed on to `discrete_scale` to control name, limits, breaks, labels and so forth.
- `start`: Gray value at low end of palette
- `end`: Gray value at high end of palette
- `na.value`: Colour to use for missing values

See Also

Other colour scales: `scale_alpha`, `scale_colour_brewer`, `scale_colour_gradient`, `scale_colour_hue`

Examples

```r
p <- ggplot(mtcars, aes(mpg, wt)) + geom_point(aes(colour = factor(cyl)))
p + scale_colour_grey()
p + scale_colour_grey(end = 0)

# You may want to turn off the pale grey background with this scale
p + scale_colour_grey() + theme_bw()

# Colour of missing values is controlled with na.value:
miss <- factor(sample(c(NA, 1:5), nrow(mtcars), replace = TRUE))
ggplot(mtcars, aes(mpg, wt)) +
  geom_point(aes(colour = miss)) +
  scale_colour_grey()
ggplot(mtcars, aes(mpg, wt)) +
  geom_point(aes(colour = miss)) +
  scale_colour_grey(na.value = "green")
```
**scale_colour_hue**

*Evenly spaced colours for discrete data*

**Description**

This is the default colour scale for categorical variables. It maps each level to an evenly spaced hue on the colour wheel. It does not generate colour-blind safe palettes.

**Usage**

```r
scale_colour_hue(..., h = c(0, 360) + 15, c = 100, l = 65, h.start = 0, direction = 1, na.value = "grey50")

scale_fill_hue(..., h = c(0, 360) + 15, c = 100, l = 65, h.start = 0, direction = 1, na.value = "grey50")
```

**Arguments**

- `...`: Other arguments passed on to `discrete_scale` to control name, limits, breaks, labels and so forth.
- `h`: range of hues to use, in [0, 360]
- `c`: chroma (intensity of colour), maximum value varies depending on combination of hue and luminance.
- `l`: luminance (lightness), in [0, 100]
- `h.start`: hue to start at
- `direction`: direction to travel around the colour wheel, 1 = clockwise, -1 = counter-clockwise
- `na.value`: Colour to use for missing values

**See Also**

Other colour scales: `scale_alpha`, `scale_colour_brewer`, `scale_colour_gradient`, `scale_colour_grey`

**Examples**

```r
dsamp <- diamonds[sample(nrow(diamonds), 1000), ]
(d <- ggplot(dsamp, aes(carat, price)) + geom_point(aes(colour = clarity)))

# Change scale label
d + scale_colour_hue()
d + scale_colour_hue("clarity")
d + scale_colour_hue(expression(clarity[beta]))

# Adjust luminosity and chroma
d + scale_colour_hue(l = 40, c = 30)
d + scale_colour_hue(l = 70, c = 30)
```
scale_continuous

Position scales for continuous data (x & y)

Description

scale_x_continuous and scale_y_continuous are the default scales for continuous x and y aesthetics. There are three variants that set the trans argument for commonly used transformations: scale_*_log10, scale_*_sqrt and scale_*_reverse.

Usage

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

d + scale_x_continuous(l = 70, c = 150)
d + scale_x_continuous(l = 80, c = 150)

# Change range of hues used
d + scale_colour_hue(h = c(0, 90))
d + scale_colour_hue(h = c(90, 180))
d + scale_colour_hue(h = c(180, 270))
d + scale_colour_hue(h = c(270, 360))

# Vary opacity
# (only works with pdf, quartz and cairo devices)
d <- ggplot(dsamp, aes(carat, price, colour = clarity))
d + geom_point(alpha = 0.9)
d + geom_point(alpha = 0.5)
d + geom_point(alpha = 0.2)

# Colour of missing values is controlled with na.value:
miss <- factor(sample(c(NA, 1:5), nrow(mtcars), replace = TRUE))
ggplot(mtcars, aes(mpg, wt)) + geom_point(aes(colour = miss))
ggplot(mtcars, aes(mpg, wt)) +
  geom_point(aes(colour = miss)) +
  scale_colour_hue(na.value = "black")
```

scale_y_continuous

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

d + scale_y_continuous(l = 70, c = 150)
d + scale_y_continuous(l = 80, c = 150)
```

scale_y_log10

```
scale_y_log10(...)```

scale_x_log10

```
scale_x_log10(...)```
scale_continuous

scale_x_reverse(...)
scale_y_reverse(...)
scale_x_sqrt(...)
scale_y_sqrt(...)

Arguments

name
The name of the scale. Used as axis or legend title. If NULL, the default, the name of the scale is taken from the first mapping used for that aesthetic.

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

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
A numeric vector of length two providing limits of the scale. Use NA to refer to the existing minimum or maximum.

expand
A numeric vector of length two giving multiplicative and additive expansion constants. These constants ensure that the data is placed some distance away from the axes. The defaults are \(c(0.05, 0)\) for continuous variables, and \(c(0, 0.6)\) 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", "exp", "identity", "log", "log10", "log1p", "log2", "logit", "probability", "probit", "reciprocal", "reverse" and "sqrt". 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. boxcox_trans. You can create your own transformation with trans_new.
position

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

sec.axis

specify a secondary axis

... Other arguments passed on to scale_(x|y)_continuous

Details

For simple manipulation of labels and limits, you may wish to use labs() and lims() instead.

See Also

sec_axis for how to specify secondary axes

Other position scales: scale_x_date, scale_x_discrete

Examples

```r
p1 <- ggplot(mpg, aes(displ, hwy)) +
  geom_point()
p1

# Manipulating the default position scales lets you:
# * change the axis labels
p1 +
  scale_x_continuous("Engine displacement (l)") +
  scale_y_continuous("Highway MPG")

# You can also use the short-cut labs().
# Use NULL to suppress axis labels
p1 + labs(x = NULL, y = NULL)

# * modify the axis limits
p1 + scale_x_continuous(limits = c(2, 6))
p1 + scale_x_continuous(limits = c(0, 10))

# you can also use the short hand functions `xlim()` and `ylim()`
p1 + xlim(2, 6)

# * choose where the ticks appear
p1 + scale_x_continuous(breaks = c(2, 4, 6))

# * add what labels they have
p1 + scale_x_continuous(
  breaks = c(2, 4, 6),
  label = c("two", "four", "six")
)

# Typically you'll pass a function to the 'labels' argument.
# Some common formats are built into the scales package:
df <- data.frame(
  x = rnorm(10) * 100000,
  y = seq(0, 1, length.out = 10)
)
\begin{verbatim}
)
p2 <- ggplot(df, aes(x, y)) + geom_point()
p2 + scale_y_continuous(labels = scales::percent)
p2 + scale_y_continuous(labels = scales::dollar)
p2 + scale_x_continuous(labels = scales::comma)

# You can also override the default linear mapping by using a
# transformation. There are three shortcuts:
p1 + scale_y_log10()
p1 + scale_y_sqrt()
p1 + scale_y_reverse()

# Or you can supply a transformation in the 'trans' argument:
p1 + scale_y_continuous(trans = scales::reciprocal_trans())

# You can also create your own. See ?scales::trans_new
\end{verbatim}

\section*{scale_date}

\textit{Position scales for date/time data}

\subsection*{Description}
These are the default scales for the three \textit{date/time} class. These will usually be added automatically. To override manually, use \texttt{scale\_\_\_date} for dates (class \texttt{Date}), \texttt{scale\_\_\_datetime} for datetimes (class \texttt{POSIXct}), and \texttt{scale\_\_\_time} for times (class \texttt{hms}).

\subsection*{Usage}

\texttt{scale\_x\_date(name = waiver(), breaks = waiver(), date_breaks = waiver(),
labels = waiver(), date_labels = waiver(), minor_breaks = waiver(),
date_minor_breaks = waiver(), limits = NULL, expand = waiver(),
position = "bottom")}

\texttt{scale\_y\_date(name = waiver(), breaks = waiver(), date_breaks = waiver(),
labels = waiver(), date_labels = waiver(), minor_breaks = waiver(),
date_minor_breaks = waiver(), limits = NULL, expand = waiver(),
position = "left")}

\texttt{scale\_x\_datetime(name = waiver(), breaks = waiver(),
date_breaks = waiver(), labels = waiver(), date_labels = waiver(),
minor_breaks = waiver(), date_minor_breaks = waiver(), timezone = NULL,
limits = NULL, expand = waiver(), position = "bottom")}

\texttt{scale\_y\_datetime(name = waiver(), breaks = waiver(),
date_breaks = waiver(), labels = waiver(), date_labels = waiver(),
minor_breaks = waiver(), date_minor_breaks = waiver(), timezone = NULL,
limits = NULL, expand = waiver(), position = "left")}

scale_date

scale_x_time(name = waiver(), breaks = waiver(), minor_breaks = waiver(),
  labels = waiver(), limits = NULL, expand = waiver(), oob = censor,
  na.value = NA_real_, position = "bottom")

scale_y_time(name = waiver(), breaks = waiver(), minor_breaks = waiver(),
  labels = waiver(), limits = NULL, expand = waiver(), oob = censor,
  na.value = NA_real_, position = "left")

Arguments

name The name of the scale. Used as axis or legend title. If NULL, the default, the
name of the scale is taken from the first mapping used for that aesthetic.

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

date_breaks A string giving the distance between breaks like "2 weeks", or "10 years". If
both breaks and date_breaks are specified, date_breaks wins.

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

date_labels A string giving the formatting specification for the labels. Codes are defined in
strftime. If both labels and date_labels are specified, date_labels wins.

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.

date_minor_breaks A string giving the distance between minor breaks like "2 weeks", or "10 years".
If both minor_breaks and date_minor_breaks are specified, date_minor_breaks wins.

limits A numeric vector of length two providing limits of the scale. Use NA to refer to
the existing minimum or maximum.

expand A numeric vector of length two giving multiplicative and additive expansion
constants. These constants ensure that the data is placed some distance away
from the axes. The defaults are c(0.05, 0) for continuous variables, and
c(0, 0.6) for discrete variables.

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

timezone
The timezone to use for display on the axes. The default (NULL) uses the timezone encoded in the data.

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.

See Also
Other position scales: scale_x_continuous, scale_x_discrete

Examples

```r
last_month <- Sys.Date() - 0:29
df <- data.frame(
  date = last_month,
  price = runif(30)
)
base <- ggplot(df, aes(date, price)) +
  geom_line()

# The date scale will attempt to pick sensible defaults for
# major and minor tick marks. Override with date_breaks, date_labels
# date_minor_breaks arguments.
base + scale_x_date(date_labels = "%b %d")
base + scale_x_date(date_breaks = "1 week", date_labels = "%W")
base + scale_x_date(date_minor_breaks = "1 day")

# Set limits
base + scale_x_date(limits = c(Sys.Date() - 7, NA))
```

scale_identity

Use values without scaling

Description

Use this set of scales when your data has already been scaled, i.e. it already represents aesthetic values that ggplot2 can handle directly. This will not produce a legend unless you also supply the breaks and labels.

Usage

scale_colour_identity(..., guide = "none")

scale_fill_identity(..., guide = "none")

scale_shape_identity(..., guide = "none")

scale_linetype_identity(..., guide = "none")
scale_alpha_identity(..., guide = "none")

scale_size_identity(..., guide = "none")

**Arguments**

... Other arguments passed on to `discrete_scale` or `continuous_scale`

guide Guide to use for this scale. Defaults to "none".

**Examples**

```r
ggplot(luv_colours, aes(u, v)) +
  geom_point(aes(colour = col), size = 3) +
  scale_color_identity() +
  coord_equal()

df <- data.frame(
  x = 1:4,
  y = 1:4,
  colour = c("red", "green", "blue", "yellow")
)
ggplot(df, aes(x, y)) + geom_tile(aes(fill = colour))
ggplot(df, aes(x, y)) +
  geom_tile(aes(fill = colour)) +
  scale_fill_identity()

# To get a legend guide, specify guide = "legend"
ggplot(df, aes(x, y)) +
  geom_tile(aes(fill = colour)) +
  scale_fill_identity(guide = "legend")

# But you'll typically also need to supply breaks and labels:
ggplot(df, aes(x, y)) +
  geom_tile(aes(fill = colour)) +
  scale_fill_identity("trt", labels = letters[1:4], breaks = df$colour,
                     guide = "legend")

# cyl scaled to appropriate size
ggplot(mtcars, aes(mpg, wt)) +
  geom_point(aes(size = cyl))

# cyl used as point size
ggplot(mtcars, aes(mpg, wt)) +
  geom_point(aes(size = cyl)) +
  scale_size_identity()
```

---

scale_linetype  
*Scale for line patterns*
scale_manual

Description

Default line types based on a set supplied by Richard Pearson, University of Manchester. Continuous values cannot be mapped to line types.

Usage

scale_linetype(..., na.value = "blank")

scale_linetype_continuous(...)

scale_linetype_discrete(..., na.value = "blank")

Arguments

... common discrete scale parameters: name, breaks, labels, na.value, limits and guide. See discrete_scale for more details

na.value The linetype to use for NA values.

Examples

base <- ggplot(economics_long, aes(date, value))
base + geom_line(aes(group = variable))
base + geom_line(aes(linetype = variable))

# See scale_manual for more flexibility

# Common line types -------------------------------
df_lines <- data.frame(
  linetype = factor(
    1:4,
    labels = c("solid", "longdash", "dashed", "dotted")
  )
)
ggplot(df_lines) +
  geom_hline(aes(linetype = linetype, yintercept = 0), size = 2) +
  scale_linetype_identity() +
  facet_grid(linetype = .) +
  theme_void(20)

Description

This allows you to specify your own set of mappings from levels in the data to aesthetic values.
Usage

```r
scale_colour_manual(..., values)
scale_fill_manual(..., values)
scale_size_manual(..., values)
scale_shape_manual(..., values)
scale_linetype_manual(..., values)
scale_alpha_manual(..., values)
```

Arguments

```r
... common discrete scale parameters: name, breaks, labels, na.value, limits and guide. See discrete_scale for more details
values a set of aesthetic values to map data values to. If this is a named vector, then the values will be matched based on the names. If unnamed, values will be matched in order (usually alphabetical) with the limits of the scale. Any data values that don’t match will be given na.value.
```

Examples

```r
p <- ggplot(mtcars, aes(mpg, wt)) +
  geom_point(aes(colour = factor(cyl)))
p + scale_colour_manual(values = c("red", "blue", "green"))

# It's recommended to use a named vector
cols <- c("8" = "red", "4" = "blue", "6" = "darkgreen", "10" = "orange")
p + scale_colour_manual(values = cols)

# As with other scales you can use breaks to control the appearance
# of the legend.
p + scale_colour_manual(values = cols)
p + scale_colour_manual(
  values = cols,
  breaks = c("4", "6", "8"),
  labels = c("four", "six", "eight")
)

# And limits to control the possible values of the scale
p + scale_colour_manual(values = cols, limits = c("4", "8"))
p + scale_colour_manual(values = cols, limits = c("4", "6", "8", "10"))
```
scale_shape maps discrete variables to six easily discernible shapes. If you have more than six levels, you will get a warning message, and the seventh and subsequence levels will not appear on the plot. Use `scale_shape_manual` to supply your own values. You can not map a continuous variable to shape.

Usage

```r
scale_shape(..., solid = TRUE)
```

Arguments

- `...`: common discrete scale parameters: name, breaks, labels, na.value, limits and guide. See `discrete_scale` for more details.
- `solid`: Should the shapes be solid, TRUE, or hollow, FALSE?

Examples

```r
dsmall <- diamonds[sample(nrow(diamonds), 100), ]

(d <- ggplot(dsmall, aes(carat, price)) + geom_point(aes(shape = cut)))

# the default
d + scale_shape(solid = TRUE)

# hollow
d + scale_shape(solid = FALSE)

# change order of levels
levels(dsmall$cut) <- c("Fair", "Good", "Very Good", "Premium", "Ideal")

# To change order of levels, change order of
# underlying factor

d + scale_shape(name = "Cut of diamond")

# Need to recreate plot to pick up new data

ggplot(dsmall, aes(price, carat)) + geom_point(aes(shape = cut))

# Show a list of available shapes

df_shapes <- data.frame(shape = 0:24)

ggplot(df_shapes, aes(0, 0, shape = shape)) +
  geom_point(aes(shape = shape), size = 5, fill = "red") +
  scale_shape_identity() +
  facet_wrap(~shape) +
  theme_void()
```
scale_size

Description

scale_size scales area. scale_radius scales radius. The size aesthetic is most commonly used for points and text, and humans perceive the area of points (not their radius), so this provides for optimal perception. scale_size_area ensures that a value of 0 is mapped to a size of 0.

Usage

scale_radius(name = waiver(), breaks = waiver(), labels = waiver(),
    limits = NULL, range = c(1, 6), trans = "identity", guide = "legend")

scale_size(name = waiver(), breaks = waiver(), labels = waiver(),
    limits = NULL, range = c(1, 6), trans = "identity", guide = "legend")

scale_size_area(..., max_size = 6)

Arguments

name The name of the scale. Used as axis or legend title. If NULL, the default, the name of the scale is taken from the first mapping used for that aesthetic.
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
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 A numeric vector of length two providing limits of the scale. Use NA to refer to the existing minimum or maximum.
range a numeric vector of length 2 that specifies the minimum and maximum size of the plotting symbol after transformation.
trans Either the name of a transformation object, or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "exp", "identity", "log", "log10", "log1p", "log2", "logit", "probability", "probit", "reciprocal", "reverse" and "sqrt". 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. boxcox_trans. You can create your own transformation with trans_new.
scale_x_discrete

Position scales for discrete data

Description

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.

Usage

scale_x_discrete(..., expand = waiver(), position = "bottom")

scale_y_discrete(..., expand = waiver(), position = "left")
Arguments

... common discrete scale parameters: name, breaks, labels, na.value, limits and guide. See discrete_scale for more details

expand a numeric vector of length two giving multiplicative and additive expansion constants. These constants ensure that the data is placed some distance away from the axes.

position The position of the axis. left or right for y axes, top or bottom for x axes

See Also

Other position scales: scale_x_continuous, scale_x_date

Examples

ggplot(diamonds, aes(cut)) + geom_bar()

# The discrete position scale is added automatically whenever you
# have a discrete position.
(d <- ggplot(subset(diamonds, carat > 1), aes(cut, clarity)) +
  geom_jitter())

d + scale_x_discrete("Cut")
d + scale_x_discrete("Cut", labels = c("Fair" = "F", "Good" = "G",
  "Very Good" = "VG","Perfect" = "P","Ideal" = "I"))

# Use limits to adjust the which levels (and in what order)
# are displayed
d + scale_x_discrete(limits = c("Fair","Ideal"))

# you can also use the short hand functions xlim and ylim
d + xlim("Fair","Ideal", "Good")
d + ylim("IF", "IF")

# See ?reorder to reorder based on the values of another variable
ggplot(mpg, aes(manufacturer, cty)) + geom_point()

# Use abbreviate as a formatter to reduce long names
ggplot(mpg, aes(reorder(manufacturer, cty), cty)) + geom_point() +
  scale_x_discrete(labels = abbreviate)
Description


Usage

seals

Format

A data frame with 1155 rows and 4 variables

References

http://www.stat.berkeley.edu/~brill/Papers/jspifinal.pdf

sec_axis  Specify a secondary axis

Description

This function is used in conjunction with a position scale to create a secondary axis, positioned opposite of the primary axis. All secondary axes must be based on a one-to-one transformation of the primary axes.

Usage

sec_axis(trans = NULL, name = waiver(), breaks = waiver(), labels = waiver())

dup_axis(trans = ~., name = derive(), breaks = derive(), labels = derive())

derive()
Arguments

- **trans**: A transformation formula
- **name**: The name of the secondary axis
- **breaks**: One of:
  - `NULL` for no breaks
  - `waiver()` for the default breaks computed by the transformation object
  - A numeric vector of positions
  - A function that takes the limits as input and returns breaks as output
- **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

Details

`sec_axis` is used to create the specifications for a secondary axis. Except for the `trans` argument, any of the arguments can be set to `derive()` which would result in the secondary axis inheriting the settings from the primary axis.

dup_axis is provided as a shorthand for creating a secondary axis that is a duplication of the primary axis, effectively mirroring the primary axis.

Examples

```r
p <- ggplot(mtcars, aes(cyl, mpg)) + geom_point()

# Create a simple secondary axis
p + scale_y_continuous(sec.axis = sec_axis(~+.10))

# Inherit the name from the primary axis
p + scale_y_continuous("Miles/gallon", sec.axis = sec_axis(~+.10, name = derive()))

# Duplicate the primary axis
p + scale_y_continuous(sec.axis = dup_axis())

# You can pass in a formula as a shorthand
p + scale_y_continuous(sec.axis = ~.^2)
```
**stat_ecdf**

**Compute empirical cumulative distribution**

**Description**

The empirical cumulative distribution function (ECDF) provides an alternative visualisation of distribution. Compared to other visualisations that rely on density (like `geom_histogram`), the ECDF doesn’t require any tuning parameters and handles both continuous and categorical variables. The downside is that it requires more training to accurately interpret, and the underlying visual tasks are somewhat more challenging.

**Usage**

```r
stat_ecdf(mapping = NULL, data = NULL, geom = "step",
    position = "identity", ..., n = NULL, pad = TRUE, 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.
- `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.
- `...` other arguments passed on to `layer`. These are often aesthetics, used to set an aesthetic to a fixed value, like `color = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.
- `n` if `NULL`, do not interpolate. If not `NULL`, this is the number of points to interpolate with.
- `pad` If `TRUE`, pad the ecdf with additional points (-Inf, 0) and (Inf, 1)
- `na.rm` If `FALSE` (the default), removes missing values with a warning. If `TRUE` silently removes missing values.
- `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.
- `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`. 
Computed variables

- **x**: x in data
- **y**: cumulative density corresponding to x

Examples

```r
df <- data.frame(
    x = c(rnorm(100, 0, 3), rnorm(100, 0, 10)),
    g = gl(2, 100)
)
ggplot(df, aes(x)) + stat_ecdf(geom = "step")

# Don't go to positive/negative infinity
ggplot(df, aes(x)) + stat_ecdf(geom = "step", pad = FALSE)

# Multiple ECDFs
ggplot(df, aes(x, colour = g)) + stat_ecdf()
```

---

**stat ellipse**  
*Compute normal confidence ellipses*

Description

The method for calculating the ellipses has been modified from `car::ellipse` (Fox and Weisberg, 2011)

Usage

```r
stat_ellipse(mapping = NULL, data = NULL, geom = "path",
position = "identity", ..., type = "t", level = 0.95, segments = 51,
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.
- **geom**: The geometric object to use display the data.
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 `color = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

type The type of ellipse. The default "t" assumes a multivariate t-distribution, and "norm" assumes a multivariate normal distribution. "euclid" draws a circle with the radius equal to `level`, representing the euclidean distance from the center. This ellipse probably won’t appear circular unless `coord_fixed()` is applied.

level The confidence level at which to draw an ellipse (default is 0.95), or, if `type="euclid"`, the radius of the circle to be drawn.

segments The number of segments to be used in drawing 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.

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`.

References


Examples

```r
ggplot(faithful, aes(waiting, eruptions)) + geom_point() + stat_ellipse()

ggplot(faithful, aes(waiting, eruptions, color = eruptions > 3)) + geom_point() + stat_ellipse()

ggplot(faithful, aes(waiting, eruptions, color = eruptions > 3)) + geom_point() + stat_ellipse(type = "norm", linetype = 2) + stat_ellipse(type = "t")

ggplot(faithful, aes(waiting, eruptions, color = eruptions > 3)) + geom_point() + stat_ellipse(type = "norm", linetype = 2) + stat_ellipse(type = "euclid", level = 3) + coord_fixed()
```
```r
ggplot(faithful, aes(waiting, eruptions, fill = eruptions > 3)) +
  stat_ellipse(geom = "polygon")
```

---

**Description**

This stat makes it easy to superimpose a function on top of an existing plot. The function is called with a grid of evenly spaced values along the x axis, and the results are drawn (by default) with a line.

**Usage**

```r
stat_function(mapping = NULL, data = NULL, geom = "path",
  position = "identity", ..., fun, xlim = NULL, n = 101, args = list(),
  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.

- **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.

- **fun**
  function to use. Must be vectorised.

- **xlim**
  Optionally, restrict the range of the function to this range.

- **n**
  number of points to interpolate along

- **args**
  list of additional arguments to pass to `fun`

- **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.

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

stat_function understands the following aesthetics (required aesthetics are in bold):

• group
• y

Computed variables

x x’s along a grid
y value of function evaluated at corresponding x

Examples

set.seed(1492)
df <- data.frame(  
x = rnorm(100)
)
x <- df$x
base <- ggplot(df, aes(x)) + geom_density()  
base + stat_function(fun = dnorm, colour = "red")
base + stat_function(fun = dnorm, colour = "red", args = list(mean = 3))

# Plot functions without data
# Examples adapted from Kohske Takahashi

# Specify range of x-axis
  ggplot(data.frame(x = c(0, 2)), aes(x)) +
  stat_function(fun = exp, geom = "line")

# Plot a normal curve
  ggplot(data.frame(x = c(-5, 5)), aes(x)) + stat_function(fun = dnorm)

# To specify a different mean or sd, use the args parameter to supply new values
  ggplot(data.frame(x = c(-5, 5)), aes(x)) +
  stat_function(fun = dnorm, args = list(mean = 2, sd = .5))

# Two functions on the same plot
  f <- ggplot(data.frame(x = c(0, 10)), aes(x))
  f + stat_function(fun = sin, colour = "red") +
  stat_function(fun = cos, colour = "blue")

# Using a custom function
  test <- function(x) {x ^ 2 + x + 20}
  f + stat_function(fun = test)
Description

The identity statistic leaves the data unchanged.

Usage

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

Arguments

mapping

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

data

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

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

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

A function will be called with a single argument, the plot data. The return
value must be a data.frame, and will be used as the layer data.

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 adjust-
ment function.

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

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.

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.

Examples

p <- ggplot(mtcars, aes(wt, mpg))
p + stat_identity()
**stat_summary_2d**

Bin and summarise in 2d (rectangle & hexagons)

Description

stat_summary_2d is a 2d variation of `stat_summary`. `stat_summary_hex` is a hexagonal variation of `stat_summary_2d`. The data are divided into bins defined by x and y, and then the values of z in each cell is are summarised with `fun`.

Usage

```r
stat_summary_2d(mapping = NULL, data = NULL, geom = "tile",
       position = "identity", ..., bins = 30, binwidth = NULL, drop = TRUE,
       fun = "mean", fun.args = list(), na.rm = FALSE, show.legend = NA,
       inherit.aes = TRUE)

stat_summary_hex(mapping = NULL, data = NULL, geom = "hex",
       position = "identity", ..., bins = 30, binwidth = NULL, drop = TRUE,
       fun = "mean", fun.args = list(), 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.

- **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.

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

- **bins**
  Numeric vector giving number of bins in both vertical and horizontal directions. Set to 30 by default.

- **binwidth**
  Numeric vector giving bin width in both vertical and horizontal directions. Overrides bins if both set.
drop  drop if the output of fun is NA.
fun  function for summary.
fun.args  A list of extra arguments to pass to fun
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.
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

• x: horizontal position
• y: vertical position
• z: value passed to the summary function

Computed variables

x,y  Location
value  Value of summary statistic.

See Also

stat_summary_hex for hexagonal summarization. stat_bin2d for the binning options.

Examples

d <- ggplot(diamonds, aes(carat, depth, z = price))
d + stat_summary_2d()

# Specifying function
d + stat_summary_2d(fun = function(x) sum(x^2))
d + stat_summary_2d(fun = var)
d + stat_summary_2d(fun = "quantile", fun.args = list(probs = 0.1))

if (requireNamespace("hexbin")) {
d + stat_summary_hex()
}
stat_summary_bin

**Summarise y values at unique/binned x**

**Description**

`stat_summary` operates on unique `x`; `stat_summary_bin` operators on binned `x`. They are more flexible versions of `stat_bin`: instead of just counting, they can compute any aggregate.

**Usage**

```r
stat_summary_bin(mapping = NULL, data = NULL, geom = "pointrange", position = "identity", ..., fun.data = NULL, fun.y = NULL, fun.ymin = NULL, fun.ymax = NULL, fun.args = list(), na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```

```r
stat_summary(mapping = NULL, data = NULL, geom = "pointrange", position = "identity", ..., fun.data = NULL, fun.y = NULL, fun.ymin = NULL, fun.ymax = NULL, fun.args = list(), 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.

- **geom**: Use to override the default connection between `geom_histogram`/`geom_freqpoly` and `stat_bin`.

- **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 `color = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

- **fun.data**: A function that is given the complete data and should return a data frame with variables `ymin`, `y`, and `ymax`.

- **fun.ymin**, **fun.y**, **fun.ymax**: Alternatively, supply three individual functions that are each passed a vector of `x`'s and should return a single number.
fun.args: Optional additional arguments passed on to the functions.

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.

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

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

- **x**
- **y**
- **group**

### Summary functions

You can either supply summary functions individually (`fun.y`, `fun.ymax`, `fun.ymin`), or as a single function (`fun.data`):

- **fun.data**: Complete summary function. Should take numeric vector as input and return data frame as output.
- **fun.ymin**: ymin summary function (should take numeric vector and return single number)
- **fun.y**: y summary function (should take numeric vector and return single number)
- **fun.ymax**: ymax summary function (should take numeric vector and return single number)

A simple vector function is easiest to work with as you can return a single number, but is somewhat less flexible. If your summary function computes multiple values at once (e.g. ymin and ymax), use `fun.data`.

If no aggregation functions are supplied, will default to `mean.se`.

### See Also

`geom_errorbar`, `geom_pointrange`, `geom_linerange`, `geom_crossbar` for geoms to display summarised data

### Examples

```r
d <- ggplot(mtcars, aes(cyl, mpg)) + geom_point()
d + stat_summary(fun.data = "mean_cl_boot", colour = "red", size = 2)
```

# You can supply individual functions to summarise the value at each x:
```r
d + stat_summary(fun.y = "median", colour = "red", size = 2, geom = "point")
d + stat_summary(fun.y = "mean", colour = "red", size = 2, geom = "point")
d + aes(colour = factor(vs)) + stat_summary(fun.y = mean, geom="line")
```
stat_summary_bin

```r
d + stat_summary(fun.y = mean, fun.ymin = min, fun.ymax = max, 
colour = "red")

d <- ggplot(diamonds, aes(cut))
d + geom_bar()
d + stat_summary_bin(aes(y = price), fun.y = "mean", geom = "bar")

# Don't use ylim to zoom into a summary plot - this throws the 
data away
p <- ggplot(mtcars, aes(cyl, mpg)) +
  stat_summary(fun.y = "mean", geom = "point")
p + ylim(15, 30)
# Instead use coord_cartesian
p + coord_cartesian(ylim = c(15, 30))

# A set of useful summary functions is provided from the Hmisc package:
stat_sum_df <- function(fun, geom="crossbar", ...) {
  stat_summary(fun.data = fun, colour = "red", geom = geom, width = 0.2, ...)
}
d <- ggplot(mtcars, aes(cyl, mpg)) + geom_point()
# The crossbar geom needs grouping to be specified when used with
# a continuous x axis.
d + stat_sum_df("mean_cl_boot", mapping = aes(group = cyl))
d + stat_sum_df("mean_sdl", mapping = aes(group = cyl))
d + stat_sum_df("median_hilow", mapping = aes(group = cyl))

# An example with highly skewed distributions:
if (require("ggplot2movies")) {
  set.seed(596)
mov <- movies[sample(nrow(movies), 1000), ]
m2 <- ggplot(mov, aes(x = factor(round(rating)), y = votes)) + geom_point()
m2 <- m2 + stat_summary(fun.data = "mean_cl_boot", geom = "crossbar", 
colour = "red", width = 0.3) + xlab("rating")
m2
# Notice how the overplotting skews off visual perception of the mean
# supplementing the raw data with summary statistics is _very_ important

# Next, we'll look at votes on a log scale.
# Transforming the scale means the data are transformed
# first, after which statistics are computed:
m2 + scale_y_log10()
# Transforming the coordinate system occurs after the
# statistic has been computed. This means we're calculating the summary on the raw data
# and stretching the geoms onto the log scale. Compare the widths of the
# standard errors.
m2 + coord_trans(y="log10")
}
```
stat_unique  

Remove duplicates

Description
Remove duplicates

Usage
stat_unique(mapping = NULL, data = NULL, geom = "point",
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.
- **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.
- **...**: other arguments passed on to `layer`. These are often aesthetics, used to set an aesthetic to a fixed value, like `color = "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.
- **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
stat_unique understands the following aesthetics (required aesthetics are in bold):
- **group**
### Examples

```r
ggplot(mtcars, aes(vs, am)) +
  geom_point(alpha = 0.1)
```

```r
ggplot(mtcars, aes(vs, am)) +
  geom_point(alpha = 0.1, stat = "unique")
```

### Description

Use `theme()` to modify individual components of a theme, allowing you to control the appearance of all non-data components of the plot. `theme()` only affects a single plot: see `theme_update()` if you want to modify the active theme, to affect all subsequent plots.

### Usage

```r
theme(line, rect, text, title, aspect.ratio, axis.title, axis.title.x,
  axis.title.x.top, axis.title.y, axis.title.y.right, axis.text, axis.text.x,
  axis.text.x.top, axis.text.y, axis.text.y.right, axis.ticks, axis.ticks.x,
  axis.ticks.y, axis.ticks.length, axis.line, axis.line.x, axis.line.y,
  legend.background, legend.margin, legend.spacing, legend.spacing.x,
  legend.spacing.y, legend.key, legend.key.size, legend.key.height,
  legend.key.width, legend.text, legend.text.align, legend.title,
  legend.title.align, legend.position, legend.direction, legend.justification,
  legend.box, legend.box.just, legend.box.margin, legend.box.background,
  legend.box.spacing, panel.background, panel.border, panel.spacing,
  panel.spacing.x, panel.spacing.y, panel.grid, panel.grid.major,
  panel.grid.minor, panel.grid.major.x, panel.grid.major.y, panel.grid.minor.x,
  panel.grid.minor.y, panel.ontop, plot.background, plot.title, plot.subtitle,
  plot.caption, plot.margin, strip.background, strip.placement, strip.text,
  strip.text.x, strip.text.y, strip.switch.pad.grid, strip.switch.pad.wrap, ...,
  complete = FALSE, validate = TRUE)
```

### Arguments

- **line**: all line elements (element_line)
- **rect**: all rectangular elements (element_rect)
- **text**: all text elements (element_text)
- **title**: all title elements: plot, axes, legends (element_text; inherits from text)
- **aspect.ratio**: aspect ratio of the panel
- **axis.title**: label of axes (element_text; inherits from text)
- **axis.title.x**: x axis label (element_text; inherits from axis.title)
- **axis.title.x.top**: x axis label on top axis (element_text; inherits from axis.title.x)
theme

axis.title.y        y axis label (element_text; inherits from axis.title)
axis.title.y.right  y axis label on right axis (element_text; inherits from axis.title.y)
axis.text           tick labels along axes (element_text; inherits from text)
axis.text.y         y axis tick labels (element_text; inherits from axis.text)
axis.text.y.right   y axis tick labels on right axis (element_text; inherits from axis.text.y)
axis.text.x         x axis tick labels (element_text; inherits from axis.text)
axis.text.x.top     x axis tick labels on top axis (element_text; inherits from axis.text.x)
axis.text.y         y axis tick labels (element_text; inherits from axis.text)
axis.text.y.right   y axis tick labels on right axis (element_text; inherits from axis.text.y)
axis.ticks          tick marks along axes (element_line; inherits from line)
axis.ticks.x        x axis tick marks (element_line; inherits from axis.ticks)
axis.ticks.y        y axis tick marks (element_line; inherits from axis.ticks)
axis.ticks.length   length of tick marks (unit)
axis.line           lines along axes (element_line; inherits from line)
axis.line.x         line along x axis (element_line; inherits from axis.line)
axis.line.y         line along y axis (element_line; inherits from axis.line)

legend.background   background of legend (element_rect; inherits from rect)
legend.margin       the margin around each legend (margin)
legend.spacing      the spacing between legends (unit)
legend.spacing.x    the horizontal spacing between legends (unit); inherits from legend.spacing
legend.spacing.y    the horizontal spacing between legends (unit); inherits from legend.spacing
legend.key          background underneath legend keys (element_rect; inherits from rect)
legend.key.size     size of legend keys (unit)
legend.key.height   key background height (unit; inherits from legend.key.size)
legend.key.width    key background width (unit; inherits from legend.key.size)
legend.text         legend item labels (element_text; inherits from text)
legend.text.align   alignment of legend labels (number from 0 (left) to 1 (right))
legend.title        title of legend (element_text; inherits from title)
legend.title.align  alignment of legend title (number from 0 (left) to 1 (right))
legend.position     the position of legends ("none", "left", "right", "bottom", "top", or two-element numeric vector)
legend.direction
  layout of items in legends ("horizontal" or "vertical")

legend.justification
  anchor point for positioning legend inside plot ("center" or two-element numeric vector) or the justification according to the plot area when positioned outside the plot

legend.box
  arrangement of multiple legends ("horizontal" or "vertical")

legend.box.just
  justification of each legend within the overall bounding box, when there are multiple legends ("top", "bottom", "left", or "right")

legend.box.margin
  margins around the full legend area, as specified using margin

legend.box.background
  background of legend area (element_rect; inherits from rect)

legend.box.spacing
  The spacing between the plotting area and the legend box (unit)

panel.background
  background of plotting area, drawn underneath plot (element_rect; inherits from rect)

panel.border
  border around plotting area, drawn on top of plot so that it covers tick marks and grid lines. This should be used with fill=NA (element_rect; inherits from rect)

panel.spacing
  spacing between facet panels (unit)

panel.spacing.x
  horizontal spacing between facet panels (unit; inherits from panel.spacing)

panel.spacing.y
  vertical spacing between facet panels (unit; inherits from panel.spacing)

panel.grid
  grid lines (element_line; inherits from line)

panel.grid.major
  major grid lines (element_line; inherits from panel.grid)

panel.grid.minor
  minor grid lines (element_line; inherits from panel.grid)

panel.grid.major.x
  vertical major grid lines (element_line; inherits from panel.grid.major)

panel.grid.major.y
  horizontal major grid lines (element_line; inherits from panel.grid.major)

panel.grid.minor.x
  vertical minor grid lines (element_line; inherits from panel.grid.minor)

panel.grid.minor.y
  horizontal minor grid lines (element_line; inherits from panel.grid.minor)

panel.ontop
  option to place the panel (background, gridlines) over the data layers. Usually used with a transparent or blank panel.background (logical)

plot.background
  background of the entire plot (element_rect; inherits from rect)
theme

plot.title plot title (text appearance) (element_text; inherits from title) left-aligned by default
plot.subtitle plot subtitle (text appearance) (element_text; inherits from title) left-aligned by default
plot.caption caption below the plot (text appearance) (element_text; inherits from title) right-aligned by default
plot.margin margin around entire plot (unit with the sizes of the top, right, bottom, and left margins)
strip.background background of facet labels (element_rect; inherits from rect)
strip.placement placement of strip with respect to axes, either "inside" or "outside". Only important when axes and strips are on the same side of the plot.
strip.text facet labels (element_text; inherits from text)
strip.text.x facet labels along horizontal direction (element_text; inherits from strip.text)
strip.text.y facet labels along vertical direction (element_text; inherits from strip.text)
strip.switch.pad.grid space between strips and axes when strips are switched (unit)
strip.switch.pad.wrap space between strips and axes when strips are switched (unit)
... additional element specifications not part of base ggplot2. If supplied validate needs to be set to FALSE.
complete set this to TRUE if this is a complete theme, such as the one returned by theme_gray(). Complete themes behave differently when added to a ggplot object. Also, when setting complete = TRUE all elements will be set to inherit from blank elements.
validate TRUE to run validate_element, FALSE to bypass checks.

Theme inheritance

Theme elements inherit properties from other theme elements. For example, axis.title.x inherits from axis.title, which in turn inherits from text. All text elements inherit directly or indirectly from text; all lines inherit from line, and all rectangular objects inherit from rect. This means that you can modify the appearance of multiple elements by setting a single high-level component.

See Also

+.gg and %+replace%, element_blank, element_line, element_rect, and element_text for details of the specific theme elements.

Examples

pl <- ggplot(mtcars, aes(wt, mpg)) +
  geom_point() +
  labs(title = "Fuel economy declines as weight increases")
pl
# Plot
```r
p1 + theme(plot.title = element_text(size = rel(2)))
p1 + theme(plot.background = element_rect(fill = "green"))
```

# Panels
```r
p1 + theme(panel.background = element_rect(fill = "white", colour = "grey50"))
p1 + theme(panel.border = element_rect(linetype = "dashed", fill = NA))
p1 + theme(panel.grid.major = element_line(colour = "black"))
p1 + theme(
    panel.grid.major.y = element_blank(),
    panel.grid.minor.y = element_blank()
)
```

# Put gridlines on top of data
```r
p1 + theme(
    panel.background = element_rect(fill = NA),
    panel.grid.major = element_line(colour = "grey50"),
    panel.ontop = TRUE
)
```

# Axes
```r
p1 + theme(axis.line = element_line(size = 3, colour = "grey80"))
p1 + theme(axis.text = element_text(colour = "blue"))
p1 + theme(axis.ticks = element_line(size = 2))
p1 + theme(axis.ticks.length = unit(.25, "cm"))
p1 + theme(axis.title.y = element_text(size = rel(1.5), angle = 90))
```

# Legend
```r
p2 <- ggplot(mtcars, aes(x, y)) +
    geom_point(aes(colour = factor(cyl), shape = factor(vs))) +
    labs(
        x = "Weight (1000 lbs)",
        y = "Fuel economy (mpg)",
        colour = "Cylinders",
        shape = "Transmission"
    )
p2
```

# Position
```r
p2 + theme(legend.position = "none")
p2 + theme(legend.justification = "top")
p2 + theme(legend.position = "bottom")
```

# Or place inside the plot using relative coordinates between 0 and 1
# legend.justification sets the corner that the position refers to
```r
p2 + theme(
    legend.position = c(.95, .95),
    legend.justification = c("right", "top"),
    legend.box.just = "right",
    legend.margin = margin(6, 6, 6, 6)
)
```
# The legend.box properties work similarly for the space around
# all the legends
p2 + theme(
  legend.box.background = element_rect(),
  legend.box.margin = margin(6, 6, 6)
)

# You can also control the display of the keys
# and the justification related to the plot area can be set
p2 + theme(legend.key = element_rect(fill = "white", colour = "black"))
p2 + theme(legend.text = element_text(size = 8, colour = "red"))
p2 + theme(legend.title = element_text(face = "bold"))

# Strips
p3 <- ggplot(mtcars, aes(wt, mpg)) +
  geom_point() +
  facet_wrap(~ cyl)
p3

p3 + theme(strip.background = element_rect(colour = "black", fill = "white"))
p3 + theme(strip.text.x = element_text(colour = "white", face = "bold"))
p3 + theme(panel.spacing = unit(1, "lines"))

---

theme_get

*Get, set, and modify the active theme*

**Description**

The current/active theme is automatically applied to every plot you draw. Use `theme_get` to get the current theme, and `theme_set` to completely override it. `theme_update` and `theme_replace` are shorthands for changing individual elements.

**Usage**

```r
theme_get()
theme_set(new)
theme_update(...)  
theme_replace(...)  
e1 %replace% e2
```
Arguments

- `new` new theme (a list of theme elements)
- `...` named list of theme settings
- `e1, e2` Theme and element to combine

Value

- `theme_set`, `theme_update`, and `theme_replace` invisibly return the previous theme so you can easily save it, then later restore it.

Adding on to a theme

- `+` and `%replace%` can be used to modify elements in themes.
- `+` updates the elements of `e1` that differ from elements specified (not NULL) in `e2`. Thus this operator can be used to incrementally add or modify attributes of a ggplot theme.
- In contrast, `%replace%` replaces the entire element; any element of a theme not specified in `e2` will not be present in the resulting theme (i.e. NULL). Thus this operator can be used to overwrite an entire theme.
- `theme_update` uses the `+` operator, so that any unspecified values in the theme element will default to the values they are set in the theme. `theme_replace` uses `%replace%` to completely replace the element, so any unspecified values will overwrite the current value in the theme with NULLs.

See Also

- `gg`

Examples

```r
p <- ggplot(mtcars, aes(mpg, wt)) + geom_point()

# Use theme_set() to completely override the current theme.
# Here we have the old theme so we can later restore it.
# Note that the theme is applied when the plot is drawn, not when it is created.
old <- theme_set(theme_bw())
p
theme_set(old)
p

# Modifying theme objects -----------------------------------------------
# You can use + and %replace% to modify a theme object.
# They differ in how they deal with missing arguments in the theme elements.
add_el <- theme_grey() +
  theme(text = element_text(family = "Times"))
```
txhousing

```
add_el$text

rep_el <- theme_grey() %> REPLACE %
theme(text = element_text(family = "Times"))
rep_el$text

# theme_update() and theme_replace() are similar except they
# apply directly to the current/active theme.
```

txhousing

**Housing sales in TX**

**Description**

Information about the housing market in Texas provided by the TAMU real estate center, [http://recenter.tamu.edu/](http://recenter.tamu.edu/).

**Usage**

txhousing

**Format**

A data frame with 8602 observations and 9 variables:

- **city** Name of MLS area
- **year,month,day** Date
- **sales** Number of sales
- **volume** Total value of sales
- **median** Median sale price
- **listings** Total active listings
- **inventory** "Months inventory": amount of time it would take to sell all current listings at current pace of sales.
Index

*Topic datasets
  diamonds, 29
economics, 30
faithful, 36
luv_colours, 126
midwest, 129
mpg, 130
msleep, 131
presidential, 138
seals, 163
txhousing, 184

*Topic hplot
  print.ggplot, 139
  +.gg, 5, 180, 183
  %+%(%+.gg), 5
  %+replace%(theme_get), 182
  %+replace%, 180
  aes, 5, 6, 7, 8, 37, 39, 42, 44, 45, 48, 50, 53, 55, 58, 60, 63, 65, 68, 69, 71, 75, 77, 80, 83, 85, 87, 89, 91, 93, 95, 97, 101, 102, 165, 166, 168, 170, 171, 173, 176
  aes_, 6, 7, 37, 39, 42, 44, 45, 48, 50, 53, 55, 58, 60, 63, 65, 68, 69, 71, 75, 77, 80, 83, 85, 87, 89, 91, 93, 95, 97, 101, 102, 165, 166, 168, 170, 171, 173, 176
  aes_colour_fill_alpha, 8
  aes_group_order, 9
  aes_linetype_size_shape, 11
  aes_position, 12
  aes_q(aes_), 7
  aes_string(aes_), 7
  annotate, 13
  annotation_custom, 14
  annotation_logticks, 15
  annotation_map, 17
  annotation_raster, 18
  arrow, 78, 127

  as_labeller, 119, 120, 122
  autoplot, 19

  bandwidth.nrd, 58
  boxcox_trans, 151, 160
  boxplot.stats, 46
  bquote, 122
  brewer.pal, 144
  bw.nrd, 56, 103

  capitalize, 119
  color(aesColour_fill_alpha), 8
  colors, 126
  colour(aesColour_fill_alpha), 8
  continuous_scale, 143, 146, 156, 161
  coord_cartesian, 20, 125
  coord_equal(coord_fixed), 21
  coord_fixed, 21
  coord_flip, 22
  coord_map, 23
  coord_polar, 25
  coord_quickmap(coord_map), 23
  coord_trans, 16, 26
  curveGrob, 95
  cut, 29
  cut_interval, 28
  cut_number, 29
  cut_number(cut_interval), 28
  cut_width(cut_interval), 28

  density, 56, 103
  derive(sec_axis), 163
  diamonds, 29
  discrete_scale, 143, 144, 148, 149, 156–159, 162
  dup_axis(sec_axis), 163
INDEX

economics, 30
economics_long (economics), 30
element_blank, 180
element_blank (margin), 126
element_line, 180
element_line (margin), 126
element_rect, 180
element_rect (margin), 126
element_text, 113, 115, 180
element_text (margin), 126
eqscplot, 21
expand_limits, 31, 43, 125
facet_grid, 31, 34, 119, 141
facet_wrap, 34, 122, 141
faithful, 36
faithfuld, 36
fill (aes_colour_fill_alpha), 8
format.ggproto (print.ggproto), 140
fortify, 17, 19, 36, 37, 40, 42, 44, 45, 48, 51, 53, 55, 58, 60, 63, 65, 68, 69, 71, 75, 77, 80, 83, 85, 87, 89, 92, 93, 95, 98, 101, 102, 105, 165, 166, 170, 171, 173, 176
fortify.lm, 37
gam, 98
gam_abline, 37
gam_area, 137
gam_area (geom_raster), 91
gam_bar, 39, 66, 92, 137
gam_bin2d, 42, 51, 59, 67, 80
gam_blank, 31, 43
gam_boxplot, 44, 70, 81, 87, 102
gam_col, 39
gam_col (geom_bar), 39
gam_contour, 48, 59
gam_count, 50, 80, 81
gam_crossbar, 52, 174
gam_curve (geom_segment), 95
gam_density, 55, 57, 102
gam_density2d, 81
gam_density2d (geom_density_2d), 57
gam_density_2d, 49, 57, 81
gam_dotplot, 59
gam_errorbar, 62, 174
gam_errorbar (geom_crossbar), 52
gam_errorbarh, 54, 62
gam_freqpoly, 56, 64
gam_hex, 67, 81
gam_histogram, 40, 41, 56, 165
gam_histogram (geom_freqpoly), 64
gam_hline (geom_abline), 37
gam_jitter, 47, 69, 80
gam_label, 71
gam_line, 38, 66, 96
gam_line (geom_path), 77
gam_linerange, 92, 174
gam_linerange (geom_crossbar), 52
gam_map, 75
gam_path, 77, 83, 84, 96
gam_point, 42, 50, 70, 80
gam_pointrange, 174
gam_pointrange (geom_crossbar), 52
gam_polygon, 19, 78, 83, 92
gam_qa, 85
gam_quantile, 47, 81, 87
gam_raster, 18, 89
gam_rect (geom_raster), 89
gam_ribbon, 84, 91
gam_rug, 93
gam_segment, 38, 78, 95, 100
gam_smooth, 54, 81, 97
gam_spoke, 96, 100
gam_step (geom_path), 77
gam_text, 135
gam_text (geom_label), 71
gam_tile, 24, 48
gam_tile (geom_raster), 89
gam_violin, 47, 56, 102, 104
gam_vline (geom_abline), 37
ggplot, 5, 19, 37, 39, 42, 44, 45, 48, 51, 53, 55, 58, 60, 63, 65, 68, 69, 71, 75, 77, 80, 83, 85, 87, 89, 91, 93, 95, 98, 101, 102, 105, 141, 165, 166, 168, 170, 171, 173, 176
ggplot2, 6
ggplot_build, 139
ggproto, 107
ggproto_parent (ggproto), 107
ggsave, 108
ggtheme, 109
ggtitle (labs), 124
glm, 99
gray.colors, 148
group (aes_group_order), 9
guide_colorbar (guide_colourbar), 112
guide_colourbar, 111, 112, 116
guide_legend, 111, 114, 115
guides, 111, 114, 116

hmisc, 118

is.ggproto (ggproto), 107

kde2d, 57

label_both (labellers), 121
label_bquote, 122, 123
label_context (labellers), 121
label_parsed (labellers), 121
label_value, 32, 34
label_value (labellers), 121
label_wrap_gen (labellers), 121
labeller, 32, 34, 119, 122, 123
labellers, 120, 121, 123
labs, 113, 115, 124, 152
layer, 13, 37, 40, 42, 44, 45, 49, 51, 53, 55, 58, 60, 63, 65, 68, 70, 72, 75, 78, 81, 83, 85, 89, 92, 93, 95, 98, 101, 103, 165, 167, 168, 170, 171, 173, 176

lims, 125, 152
linetype (aes_linetype_size_shape), 11
lm, 99
loess, 98, 99
luv_colours, 126

map, 19
mapproject, 23
margin, 126, 127, 179
mean_cl_boot (hmisc), 118
mean_cl_normal (hmisc), 118
mean sdl (hmisc), 118
mean_se, 128, 174
median_hilow (hmisc), 118
midwest, 129
mpg, 130
msleep, 131

plot, 141
plot.ggplot (print.ggplot), 139
png, 108
position_dodge, 40, 41, 132, 133–135, 137
position_fill, 40
position_fill (position_stack), 136
position_identity, 132, 133, 134, 135, 137
position_jitter, 132, 133, 133, 135, 137
position_jitterdodge, 132–134, 134, 135, 137
position_nudge, 132–135, 135, 137
position_stack, 40, 92, 132–135, 136
predict, 98
presidential, 138
print.ggplot, 139
print.ggproto, 140

qplot, 141
quickplot (qplot), 141

rel (margin), 126
rescale, 144, 147
resolution, 142
rq, 88

scale_alpha, 143, 145, 147–149
scale_alpha_continuous (scale_alpha), 143
scale_alpha_discrete (scale_alpha), 143
scale_alpha_identity (scale_identity), 155
scale_alpha_manual (scale_manual), 157
scale_color_brewer (scale_colour_brewer), 144
scale_color_continuous (scale_colour_gradient), 146
scale_color_discrete (scale_colour_hue), 149
scale_color_distiller (scale_colour_brewer), 144
scale_color_gradient (scale_colour_gradient), 146
scale_color_gradient2 (scale_colour_gradient), 146
scale_color_gradientn (scale_colour_gradient), 146
scale_color_grey (scale_colour_grey), 148
scale_color_hue (scale_colour_hue), 149
scale_color_identity (scale_identity), 155
scale_color_manual (scale_manual), 157
scale_colour_brewer, 143, 144, 147–149
scale_colour_continuous (scale_colour_gradient), 146
scale_colour_date
(scale_colour_gradient), 146
scale_colour_datetime
(scale_colour_gradient), 146
scale_colour_discrete
(scale_colour_hue), 149
scale_colour_distiller
(scale_colour_brewer), 144
scale_colour_gradient, 143, 145, 146, 148, 149
scale_colour_gradient2
(scale_colour_gradient), 146
scale_colour_gradientn
(scale_colour_gradient), 146
scale_colour_grey, 143, 145, 147, 148, 149
scale_colour_hue, 143, 145, 147, 148, 149
scale_colour_identity(scale_identity), 155
scale_colour_manual (scale_manual), 157
scale_continuous, 150
scale_date, 153
scale_fill_brewer
(scaleColour_brewer), 144
scale_fill_continuous
(scale_colour_gradient), 146
scale_fill_date
(scale_colour_gradient), 146
scale_fill_datetime
(scale_colour_gradient), 146
scale_fill_discrete (scale_colour_hue), 149
scale_fill_distiller
(scale_colour_brewer), 144
scale_fill_gradient
(scale_colour_gradient), 146
scale_fill_gradient2
(scale_colour_gradient), 146
scale_fill_gradientn
(scale_colour_gradient), 146
scale_fill_grey (scale_colour_grey), 148
scale_fill_hue (scale_colour_hue), 149
scale_fill_identity (scale_identity), 155
scale_fill_manual (scale_manual), 157
scale_identity, 155
scale_linetype, 156
scale_linetype_continuous
(scale_linetype), 156
scale_linetype_discrete
(scale_linetype), 156
scale_linetype_identity
(scale_identity), 155
scale_linetype_manual (scale_manual), 157
scale_manual, 157
scale_radius (scale_size), 160
scale_shape, 159
scale_shape_continuous (scale_shape), 159
scale_shape_discrete (scale_shape), 159
scale_shape_identity (scale_identity), 155
scale_shape_manual, 159
scale_shape_manual (scale_manual), 157
scale_size, 160
scale_size_area, 161
scale_size_area (scale_size), 160
scale_size_continuous (scale_size), 160
scale_size_date (scale_size), 160
scale_size_datetime (scale_size), 160
scale_size_discrete (scale_size), 160
scale_size_identity (scale_identity), 155
scale_size_manual (scale_manual), 157
scale_x_continuous, 155, 162
scale_x_continuous (scale_continuous), 150
scale_x_date, 152, 162
scale_x_date (scale_date), 153
scale_x_datetime (scale_date), 153
scale_x_discrete, 152, 155, 161
scale_x_log10 (scale_continuous), 150
scale_x_reverse (scale_continuous), 150
scale_x_sort (scale_continuous), 150
scale_x_time (scale_date), 153
scale_y_continuous, 16
scale_y_continuous (scale_continuous), 150
scale_y_date (scale_date), 153
scale_y_datetime (scale_date), 153
scale_y_discrete (scale_x_discrete), 161
scale_y_log10, 16
scale_y_log10 (scale_continuous), 150
scale_y_reverse (scale_continuous), 150
scale_y_sort (scale_continuous), 150
scale_y_time (scale_date), 153
seals, 163
sec_axis, 152, 163
seq_gradient_pal, 147
shape (aes_linetype_size_shape), 11
size (aes_linetype_size_shape), 11
smean.cl.boot, 118
smean.cl.normal, 118
smean.sd, 118
smedian.hilow, 118
stat_bin, 41, 66, 173
stat_bin (geom_freqpoly), 64
stat_bin2d, 69, 172
stat_bin2d (geom_bin2d), 42
stat_bin2d (geom_bin2d), 42
stat_bin_hex (geom_hex), 67
stat_binhex, 43
stat_binhex (geom_hex), 67
stat_boxplot (geom_boxplot), 44
stat_contour, 59
stat_contour (geom_contour), 48
stat_count, 64, 66
stat_count (geom_bar), 39
stat_density, 104
stat_density (geom_density), 55
stat_density2d (geom_density_2d), 57
stat_density2d (geom_density_2d), 57
stat_ecdf, 165
stat_ellipse, 166
stat_function, 168
stat_identity, 170
stat_qq (geom_qq), 85
stat_quantile (geom_quantile), 87
stat_smooth (geom_smooth), 97
stat_spoke (geom_spoke), 100
stat_sum (geom_count), 50
stat_summary, 54, 118, 128, 171
stat_summary (stat_summary_bin), 173
stat_summary2d (stat_summary_2d), 171
stat_summary_2d, 171, 171
stat_summary_bin, 173
stat_summary_hex, 172
stat_summary_hex (stat_summary_2d), 171
stat_unique, 176
stat_ydensity (geom_violin), 102
strftm, 154
strwrap, 122
substitute, 7
theme, 5, 6, 109, 113, 115, 116, 126, 177
theme_bw (ggtheme), 109
theme_classic (ggtheme), 109
theme_dark (ggtheme), 109
theme_get, 182
theme_gray (ggtheme), 109
theme_grey (ggtheme), 109
theme_light (ggtheme), 109
theme_linedraw (ggtheme), 109
theme_minimal (ggtheme), 109
theme_replace (theme_get), 182
theme_set (theme_get), 182
theme_update, 177
theme_update (theme_get), 182
theme_void (ggtheme), 109
txhousing, 184
unit, 16, 113, 116
waiver, 113, 115
x (aes_position), 12
xend (aes_position), 12
xlab (labs), 124
xlim (lims), 125
xmax (aes_position), 12
xmin (aes_position), 12
y (aes_position), 12
yend (aes_position), 12
ylab (labs), 124
ylim (lims), 125
ymax (aes_position), 12
ymin (aes_position), 12