Package ‘ggplot2’

July 3, 2018

Version 3.0.0

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), lazyeval, MASS, mgcv, plyr (>= 1.7.1), reshape2, rlang, scales (>= 0.5.0), stats, tibble, viridisLite, withr (>= 2.0.0)

Suggests covr, dplyr, ggplot2movies, hexbin, Hmisc, lattice, mapproj, maps, maptools, multcomp, munsell, nlme, testthat (>= 0.11.0), vdiffr, quantreg, knitr, rgeos, rpart, rmarkdown, sf (>= 0.3-4), svglite (>= 1.2.0.9001)

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' 'autolayer.r' 'autoplot.r' 'axis-secondary.R'
'backports.R' 'bench.r' 'bin.R' 'compat-quosures.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'
VignetteBuilder  knitr
RoxygenNote  6.0.1.9000
Encoding  UTF-8
NeedsCompilation  no
Author  Hadley Wickham [aut, cre],
        Winston Chang [aut],
        Lionel Henry [aut],
Thomas Lin Pedersen [aut],
Kohske Takahashi [aut],
Claus Wilke [aut],
Kara Woo [aut],
RStudio [cph]

Maintainer Hadley Wickham <hadley@rstudio.com>

Repository CRAN

Date/Publication 2018-07-03 17:20:03 UTC

R topics documented:

+gg .................................................. 5
aes ................................................. 7
aes_ .............................................. 8
aes_colour_fill_alpha ............................. 10
aes_group_order ................................. 11
aes_linetype_size_shape ......................... 12
aes_position ................................... 14
annotate ......................................... 15
annotation_custom ............................... 16
annotation_logticks ............................ 17
annotation_map ................................ 19
annotation_raster .............................. 19
autolayer ........................................ 20
autoplot .......................................... 21
borders .......................................... 21
coord_cartesian ................................ 23
coord_fixed ..................................... 24
coord_flip ...................................... 25
coord_map ....................................... 26
coord_polar ..................................... 28
coord_trans ..................................... 30
cut_interval .................................... 32
diamonds ........................................ 33
economics ....................................... 34
expand_limits ................................... 34
expand_scale .................................... 35
facet_grid ....................................... 36
facet_wrap ....................................... 38
faithful ......................................... 40
fortify .......................................... 40
geom_abline .................................... 41
geom_bar ........................................ 43
geom_bin2d ..................................... 46
geom_blank ..................................... 48
geom_boxplot .................................. 49
geom_contour .................................. 52
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>geom_count . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>54</td>
</tr>
<tr>
<td>geom_crossbar . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>57</td>
</tr>
<tr>
<td>geom_density . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>59</td>
</tr>
<tr>
<td>geom_density_2d . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>62</td>
</tr>
<tr>
<td>geom_dotplot . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>64</td>
</tr>
<tr>
<td>geom_errorbarh . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>67</td>
</tr>
<tr>
<td>geom_freqpoly . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>69</td>
</tr>
<tr>
<td>geom_hex . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>72</td>
</tr>
<tr>
<td>geom_jitter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>74</td>
</tr>
<tr>
<td>geom_label . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>76</td>
</tr>
<tr>
<td>geom_map . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>80</td>
</tr>
<tr>
<td>geom_path . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>82</td>
</tr>
<tr>
<td>geom_point . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>85</td>
</tr>
<tr>
<td>geom_polygon . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>88</td>
</tr>
<tr>
<td>geom_qq_line . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>90</td>
</tr>
<tr>
<td>geom_quantile . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>92</td>
</tr>
<tr>
<td>geom_raster . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>94</td>
</tr>
<tr>
<td>geom_ribbon . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>97</td>
</tr>
<tr>
<td>geom_rug . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>99</td>
</tr>
<tr>
<td>geom_segment . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>101</td>
</tr>
<tr>
<td>geom_smooth . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>103</td>
</tr>
<tr>
<td>geom_spoke . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>107</td>
</tr>
<tr>
<td>geom_violin . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>109</td>
</tr>
<tr>
<td>ggplot . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>112</td>
</tr>
<tr>
<td>ggproto . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>113</td>
</tr>
<tr>
<td>ggsave . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>115</td>
</tr>
<tr>
<td>ggsf . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>116</td>
</tr>
<tr>
<td>ggtheme . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>119</td>
</tr>
<tr>
<td>guides . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>120</td>
</tr>
<tr>
<td>guide_colourbar . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>122</td>
</tr>
<tr>
<td>guide_legend . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>125</td>
</tr>
<tr>
<td>hmisc . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>128</td>
</tr>
<tr>
<td>labeller . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>129</td>
</tr>
<tr>
<td>labellers . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>131</td>
</tr>
<tr>
<td>label_bquote . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>133</td>
</tr>
<tr>
<td>labs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>134</td>
</tr>
<tr>
<td>lims . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>135</td>
</tr>
<tr>
<td>luv_colours . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>136</td>
</tr>
<tr>
<td>margin . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>136</td>
</tr>
<tr>
<td>mean_se . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>138</td>
</tr>
<tr>
<td>midwest . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>139</td>
</tr>
<tr>
<td>mpg . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>140</td>
</tr>
<tr>
<td>msleep . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>141</td>
</tr>
<tr>
<td>position_dodge . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>142</td>
</tr>
<tr>
<td>position.identity . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>143</td>
</tr>
<tr>
<td>position_jitter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>144</td>
</tr>
<tr>
<td>position_jitterdodge . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>145</td>
</tr>
<tr>
<td>position.nudge . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .</td>
<td>146</td>
</tr>
</tbody>
</table>
### Index

<table>
<thead>
<tr>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add components to a plot</td>
<td>210</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:

- An `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 the current coordinate system.
- A `facet` specification overrides the 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())
```
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
`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 standardises 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`).

Value
A list with class `uneval`. Components of the list are either quosures or constants.

Quasiquotation
`aes()` is a quoting function. This means that its inputs are quoted to be evaluated in the context of the data. This makes it easy to work with variables from the data frame because you can name those directly. The flip side is that you have to use quasiquotation to program with `aes()`. See a tidy evaluation tutorial such as the dplyr programming vignette to learn more about these techniques.

See Also
`vars()` for another quoting function designed for faceting specifications.

Examples
```r
aes(x = mpg, y = wt)
aes(mpg, wt)

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

# Or to constants
aes(x = 1, colour = "smooth")```
# Aesthetic names are automatically standardised
aes(col = x)
aes(fg = x)
aes(color = x)
aes(colour = x)

# aes() is passed to either ggplot() or specific layer. Aesthetics supplied
# to ggplot() are used as defaults for every layer.
ggplot(mpg, aes(displ, hwy)) + geom_point()
ggplot(mpg) + geom_point(aes(displ, hwy))

# Tidy evaluation
# aes() automatically quotes all its arguments, so you need to use tidy
# evaluation to create wrappers around ggplot2 pipelines. The simplest case occurs when your wrapper takes dots:
scatter_by <- function(data, ...) {
ggplot(data) + geom_point(aes(...))
}
scatter_by(mtcars, disp, drat)

# If your wrapper has a more specific interface with named arguments,
# you need "enquote and unquote":
scatter_by <- function(data, x, y) {
x <- enquo(x)
y <- enquo(y)

ggplot(data) + geom_point(aes(!x, !y))
}
scatter_by(mtcars, disp, drat)

# Note that users of your wrapper can use their own functions in the quoted expressions and all will resolve as it should!
cut3 <- function(x) cut_number(x, 3)
scatter_by(mtcars, cut3(disp), drat)

---

**aes_**

**Define aesthetic mappings programatically**

**Description**

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 "" 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.

Life cycle

All these functions are soft-deprecated. Please use tidy evaluation idioms instead (see the quasiquotation section in aes() documentation).

See Also

aes()

Examples

# 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
c + geom_bar(fill = "red")
# Compare with the colour aesthetic which changes just the bar outline
# Combining both, you can see the changes more clearly
c + geom_bar(colour = "red")
c + 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
k <- ggplot(mtcars, aes(factor(cyl), fill = factor(vs)))
k + geom_bar()

# Fill aesthetic can also be used with a continuous variable
m <- ggplot(faithfuld, 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
```
# Aesthetics: grouping

## 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
```r
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
```r
rescale01 <- function(x) (x - min(x)) / diff(range(x))
ec_scaled <- data.frame(  
  date = economics$date,
  plyr::colwise(rescale01)(economics[, -(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, stat(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")

---

**aes_linetype_size_shape**

*Differentiation related aesthetics: linetype, size, shape*

**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, value)) + 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)
Description

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

Examples

```r
# 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
p <- ggplot(mtcars, aes(wt, mpg)) + geom_point()
p + annotate("rect", xmin = 2, xmax = 3.5, ymin = 2, ymax = 25,
                fill = "dark grey", alpha = .5)

# Geom_segment examples
p + geom_segment(aes(x = 2, y = 15, xend = 2, yend = 25),
                 arrow = arrow(length = unit(0.5, "cm")))
p + geom_segment(aes(x = 2, y = 15, xend = 3, yend = 15),
                 arrow = arrow(length = unit(0.5, "cm")))
p + geom_segment(aes(x = 5, y = 30, xend = 3.5, yend = 25),
                 arrow = arrow(length = unit(0.5, "cm")))

# You can also use geom_segment to recreate plot(type = "h") :
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))
ggplot(counts, aes(x, Freq)) +
       geom_segment(aes(yend = 0, xend = x), size = 10)
```
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
```
p <- ggplot(mtcars, aes(x = wt, y = mpg)) + geom_point()
p + annotate("text", x = 4, y = 25, label = "Some text")
p + annotate("text", 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, ymin = 12, ymax = 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",
```
annotation_custom

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

```r
# 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

def2 <- 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.
- **scaled**: 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 `grid::unit()` object specifying the length of the short tick marks
- **mid**: a `grid::unit()` object specifying the length of the middle tick marks. In base 10, these are the "5" ticks.
- **long**: a `grid::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

```r
# 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))

  ggplot(seal2, 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(latr ~ longr, 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

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

# 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
ggplot(mtcars, aes(mpg, wt)) +
  annotation_raster(rainbow, -Inf, Inf, -Inf, Inf) +
  geom_point()

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()

Description

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

Usage

autolayer(object, ...)

Arguments

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

Value
a ggplot layer

See Also
autoplot(), ggplot() and fortify()

___________________________

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
autolayer(), ggplot() and fortify()

___________________________

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 maps::map() for details
regions   map region
fill      fill colour
colour    border colour
xlim, ylim latitudinal and longitudinal range for extracting map polygons, see maps::map() for details.
...

Arguments passed on to geom_polygon

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

Examples

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)
coord_cartesian

```r
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()
```

---

**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,
               default = FALSE, clip = "on")
```

**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`.
- `default` Is this the default coordinate system? If FALSE (the default), then replacing this coordinate system with another one creates a message alerting the user that the coordinate system is being replaced. If TRUE, that warning is suppressed.
- `clip` Should drawing be clipped to the extent of the plot panel? A setting of "on" (the default) means yes, and a setting of "off" means no. In most cases, the default of "on" should not be changed, as setting `clip = "off"` can cause unexpected results. It allows drawing of data points anywhere on the plot, including in the plot margins. If limits are set via `xlim` and `ylim` and some data points fall outside those limits, then those data points may show up in places such as the axes, the legend, the plot title, or the plot margins.
Examples

# There are two ways of zooming the plot display: with scales or
# with coordinate systems. They work in two rather different ways.

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 `MASS::eqscplot()`, but it works for all types of graphics.
**Usage**

```r
coord_fixed(ratio = 1, xlim = NULL, ylim = NULL, expand = TRUE, clip = "on")
```

**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`.
- `clip`: Should drawing be clipped to the extent of the plot panel? A setting of "on" (the default) means yes, and a setting of "off" means no. In most cases, the default of "on" should not be changed, as setting `clip = "off"` can cause unexpected results. It allows drawing of data points anywhere on the plot, including in the plot margins. If limits are set via `xlim` and `ylim` and some data points fall outside those limits, then those data points may show up in places such as the axes, the legend, the plot title, or the plot margins.

**Examples**

```r
# 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(xlim = c(15, 30))

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

---

**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**

```r
coord_flip(xlim = NULL, ylim = NULL, expand = TRUE, clip = "on")
```
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.
clip       Should drawing be clipped to the extent of the plot panel? A setting of "on" (the default) means yes, and a setting of "off" means no. In most cases, the default of "on" should not be changed, as setting clip = "off" can cause unexpected results. It allows drawing of data points anywhere on the plot, including in the plot margins. If limits are set via xlim and ylim and some data points fall outside those limits, then those data points may show up in places such as the axes, the legend, the plot title, or the plot margins.

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

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()

coord_map  Map projections

description

coord_map projects a portion of the earth, which is approximately spherical, onto a flat 2D plane using any projection defined by the mapproj package. Map projections do not, in general, preserve straight lines, so this requires considerable computation. coord_quickmap is a quick approximation that does preserve straight lines. It works best for smaller areas closer to the equator.
coord_map

Usage

coord_map(projection = "mercator", ..., parameters = NULL,
orIENTATION = NULL, xlim = NULL, ylim = NULL, clip = "on")

coord_quickmap(xlim = NULL, ylim = NULL, expand = TRUE, clip = "on")

Arguments

projection    projection to use, see mapproj::mapproject() for list
... , parameters    Other arguments passed on to mapproj::mapproject(). Use ... for named
                    parameters to the projection, and parameters for unnamed parameters. ... is
                    ignored if the parameters argument is present.
orientation    projection orientation, which defaults to \(c(90, 0, \text{mean}(\text{range}(x)))\). This
                    is not optimal for many projections, so you will have to supply your own. See
                    mapproj::mapproject() for more information.
xlim, ylim    Manually specific x/y limits (in degrees of longitude/latitude)
clip    Should drawing be clipped to the extent of the plot panel? A setting of "on"
                    (the default) means yes, and a setting of "off" means no. For details, please see
                    coord_cartesian().
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 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 appro-
priate 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

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
nzm <- coord_map("cylindrical")
nzm + coord_map("azimuthal", orientation = c(-36.92, 174.6, 0))
nzm + coord_map("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_map("gilbert")
usamap + coord_map("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))
#

cord_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, clip = "on")

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
clip Should drawing be clipped to the extent of the plot panel? A setting of "on" (the default) means yes, and a setting of "off" means no. For details, please see coord_cartesian().

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
xc <- ggplot(mtcars, aes(x = factor(cyl))) +
  geom_bar(width = 1, colour = "black")
xc + 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 = "Pac man")

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

Arguments

- x, y transformers for x and y axes
- limx, limy limits for x and y axes. (Named so for backward compatibility)
- clip Should drawing be clipped to the extent of the plot panel? A setting of "on" (the default) means yes, and a setting of "off" means no. For details, please see coord_cartesian().
- xtrans, ytrans Deprecated; use x and y instead.

Details
Transformations only work with continuous values: see scales::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
coord_trans

```r
# 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)

data frame `df`:

df <- data.frame(a = abs(rnorm(26)), letters)

# Also works with discrete scales
**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 \( \text{width} \).

**Usage**

```r
cut_interval(x, n = NULL, length = NULL, ...)
cut_number(x, n = NULL, ...)
cut_width(x, width, center = NULL, boundary = NULL, closed = c("right", "left"))
```

**Arguments**

- \( x \) numeric vector
- \( n \) number of intervals to create, OR \( \text{length} \) length of each interval
- `...` Arguments passed on to `base::cut.default`
  - **breaks** either a numeric vector of two or more unique cut points or a single number (greater than or equal to 2) giving the number of intervals into which \( x \) is to be cut.
  - **labels** labels for the levels of the resulting category. By default, labels are constructed using "\( [a,b] \)" interval notation. If `labels = FALSE`, simple integer codes are returned instead of a factor.
  - **right** logical, indicating if the intervals should be closed on the right (and open on the left) or vice versa.
  - **dig.lab** integer which is used when labels are not given. It determines the number of digits used in formatting the break numbers.
  - **ordered_result** logical: should the result be an ordered factor?

- `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.

Examples

```r
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), SI2, SI1, VS2, VS1, VVS2, VVS1, 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 = \( z / \text{mean}(x, y) = 2 \times z / (x + y) \) (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](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](http://research.stlouisfed.org/fred2/series/PSAVERT)
- **pce**: personal consumption expenditures, in billions of dollars, [http://research.stlouisfed.org/fred2/series/PCE](http://research.stlouisfed.org/fred2/series/PCE)
- **unemploy**: number of unemployed in thousands, [http://research.stlouisfed.org/fred2/series/UNEMPLOY](http://research.stlouisfed.org/fred2/series/UNEMPLOY)
- **uempmed**: median duration of unemployment, in weeks, [http://research.stlouisfed.org/fred2/series/UEMPMED](http://research.stlouisfed.org/fred2/series/UEMPMED)
- **pop**: total population, in thousands, [http://research.stlouisfed.org/fred2/series/POP](http://research.stlouisfed.org/fred2/series/POP)

**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**

- `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, 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)))
```

---

**expand_scale**

Generate expansion vector for scales.

**Description**

This is a convenience function for generating scale expansion vectors for the `expand` argument of `scale_*_continuous` and `scale_*_discrete`. The expansions vectors are used to add some space between the data and the axes.

**Usage**

```r
expand_scale(mult = 0, add = 0)
```

**Arguments**

- `mult` vector of multiplicative range expansion factors. If length 1, both the lower and upper limits of the scale are expanded outwards by `mult`. If length 2, the lower limit is expanded by `mult[1]` and the upper limit by `mult[2]`.
- `add` vector of additive range expansion constants. If length 1, both the lower and upper limits of the scale are expanded outwards by `add` units. If length 2, the lower limit is expanded by `add[1]` and the upper limit by `add[2]`.

**Examples**

```r
# No space below the bars but 10% above them
ggplot(mtcars) +
  geom_bar(aes(x = factor(cyl))) +
  scale_y_continuous(expand = expand_scale(mult = c(0, .1)))

# Add 2 units of space on the left and right of the data
ggplot(subset(diamonds, carat > 2), aes(cut, clarity)) +
  geom_jitter() +
  scale_x_discrete(expand = expand_scale(add = 2))

# Reproduce the default range expansion used
```
# when the 'expand' argument is not specified
ggplot(subset(diamonds, carat > 2), aes(cut, price)) +
  geom_jitter() +
  scale_x_discrete(expand = expand_scale(add = .6)) +
  scale_y_continuous(expand = expand_scale(mult = .05))

---

**facet_grid**

**Lay out panels in a grid**

**Description**

*facet_grid* forms a matrix of panels defined by row and column faceting 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(rows = NULL, cols = NULL, scales = "fixed", space = "fixed",
          shrink = TRUE, labeller = "label_value", as.table = TRUE,
          switch = NULL, drop = TRUE, margins = FALSE, facets = NULL)
```

**Arguments**

**rows, cols**

A set of variables or expressions quoted by *vars()* and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to *labeller*).

For compatibility with the classic interface, *rows* can also be 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).

**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()*.

**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_grid**

**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.

**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.

**facets**
This argument is soft-deprecated, please use rows and cols instead.

**Examples**

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

# Use vars() to supply variables from the dataset:
p + facet_grid(rows = vars(drv))
p + facet_grid(cols = vars(cyl))
p + facet_grid(vars(drv), vars(cyl))

# The historical formula interface is also available:
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 faceted dataset with a dataset that lacks those
# faceting variables, the data will be repeated across the missing
# combinations:
df <- data.frame(displ = mean(mpg$displ), cty = mean(mpg$cty))
p +
  facet_grid(cols = vars(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
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

facet_wrap(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 A set of variables or expressions quoted by vars() and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to labeller).

For compatibility with the classic interface, can also be a formula or character vector. Use either a one sided formula, `-a` • b, or a character vector, c("a", "b").

nrow, ncol Number of rows and columns.

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

shrink If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.
labeller A function that takes one data frame of labels and returns a list or data frame of character vectors. Each input column corresponds to one factor. Thus there will be more than one with formulae of the type ~cyl + am. Each output column gets displayed as one separate line in the strip label. This function should inherit from the "labeller" S3 class for compatibility with `labeller()`. See `label_value()` for more details and pointers to other options.

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

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

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

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

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

Examples

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

# Use vars() to supply faceting variables:
p + facet_wrap(vars(class))

# The historical interface with formulas is also available:
p + facet_wrap(~class)

# Control the number of rows and columns with nrow and ncol
p + facet_wrap(vars(class), nrow = 4)

# You can facet by multiple variables
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  facet_wrap(vars(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 faceting variable.
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.
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")

faithfuld  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.
**geom_abline**

**Usage**

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

**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()`

---

**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.

- `na.rm` If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
- `show.legend` logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
- `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 differently from 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 = 1)`, 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
```

geom_bar

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().
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.
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 to 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. It
can also be a named logical vector to finely select the aesthetics to display.
inherit.aes  If FALSE, overrides the default aesthetics, rather than combining with them.
This is most useful for helper functions that define both data and aesthetics and
shouldn’t inherit behaviour from the default plot specification, e.g. borders().
geom, 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 bars 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() or position_dodge2().
Finally, position_fill() shows relative proportions at each x by stacking the bars and then stan-
dardising each bar to have the same height.

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

• x
• y
• alpha
• colour
• fill
geom_bar

• group
• linetype
• size

Learn more about setting these aesthetics in vignette("ggplot2-specs")

Computed variables

count  number of points in bin

prop  groupwise proportion

See Also
g geom_histogram() for continuous data, position_dodge() and position_dodge2() 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

# 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))

# Bar charts are automatically stacked when multiple bars are placed
# at the same location. The order of the fill is designed to match
# the legend

g + geom_bar(aes(fill = drv))

# If you need to flip the order (because you've flipped the plot)
# call position_stack() explicitly:
g +
geom_bar(aes(fill = drv), position = position_stack(reverse = TRUE)) +
coord_flip() +
theme(legend.position = "top")

# To show (e.g.) means, you need 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)

---

**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.
geom_bin2d

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

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

gcuf, stat  Use to override the default connection between geom_bin2d and stat_bin2d.

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  if TRUE removes all cells with 0 counts.

Aesthetics

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

• x
• y
• fill
• group

Learn more about setting these aesthetics in vignette("ggplot2-specs")

See Also

stat_binhex() for hexagonal binning

Examples

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**

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

**Examples**

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

```r
gem_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.
Sometimes it can be useful to hide the outliers, for example when overlaying the raw data points on top of the boxplot. Hiding the outliers can be achieved by setting `outlier.shape = NA`. Importantly, this does not remove the outliers, it only hides them, so the range calculated for the y-axis will be the same with outliers shown and outliers hidden.

**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. It can also be a named logical vector to finely select the aesthetics to display.

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

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

**coef**
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 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 * IQR / sqrt(n). This gives a roughly 95% confidence interval for comparing medians. See McGill et al. (1978) for more details.

### Aesthetics
`geom_boxplot` understands the following aesthetics (required aesthetics are in bold):

- `x`
- `lower`
- `upper`
- `middle`
- `ymin`
geom_boxplot

- ymax
- alpha
- colour
- fill
- group
- linetype
- shape
- size
- weight

Learn more about setting these aesthetics in vignette("ggplot2-specs")

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 - 1.5 * IQR / sqrt(n)
- middle median, 50% quantile
- notchupper upper edge of notch = median + 1.5 * 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":
y <- rnorm(100)
df <- data.frame(
  x = 1,
  y0 = min(y),
  y25 = quantile(y, 0.25),
  y50 = median(y),
  y75 = quantile(y, 0.75),
  y100 = max(y)
)

## It's possible to draw a boxplot with your own computations if you
## use stat = "identity":
y <- rnorm(100)
df <- data.frame(
  x = 1,
  y0 = min(y),
  y25 = quantile(y, 0.25),
  y50 = median(y),
  y75 = quantile(y, 0.75),
  y100 = max(y)
)

ggplot(df, aes(x)) +
  geom_boxplot(
    aes(ymin = y0, lower = y25, middle = y50, upper = y75, ymax = y100),
    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**

```r
geom_contour(mapping = NULL, data = NULL, stat = "contour",
  position = "identity", ..., lineend = "butt", linejoin = "round",
  linemitre = 10, na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```
geom_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 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).

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

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

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

gem 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

Learn more about setting these aesthetics in vignette("ggplot2-specs")

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 = stat(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 `geom_point()` that counts the number of observations at each location, then maps the count to point area. It 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.
- **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. It can also be a named logical vector to finely select the aesthetics to display.
- **inherit.aes**: If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders()`.
- **geom, 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

Learn more about setting these aesthetics in vignette("ggplot2-specs")

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()

# 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 = stat(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 = stat(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 = stat(prop), group = cut)) +
  scale_size_area(max_size = 10)
d + geom_count(aes(size = stat(prop), group = clarity)) +
  scale_size_area(max_size = 10)
```
Description

Various ways of representing a vertical interval defined by \( x, \) \( y_{\text{min}} \) and \( y_{\text{max}} \). 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)
```

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

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

```r
geom_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`.
A multiplicative factor used to increase the size of the middle bar in `geom_crossbar()` and the middle point in `geom_pointrange()`.

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

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

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

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

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

Learn more about setting these aesthetics in `vignette("ggplot2-specs")`

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 +
g.geom_line(aes(group = group)) +
  g.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 +
g.geom_col(position = "dodge") +
  g.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 +
g.geom_col(position = dodge) +
  g.geom_errorbar(aes(ymin = lower, ymax = upper), position = dodge, width = 0.25)

# When using g.geom_errorbar() with position_dodge2(), extra padding will be
# needed between the error bars to keep them aligned with the bars.
p +
g.geom_col(position = "dodge2") +
  g.geom_errorbar(
      aes(ymin = lower, ymax = upper),
      position = position_dodge2(width = 0.5, padding = 0.5)
  )

---

**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 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. It can also be a named logical vector to finely select the aesthetics to display.

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

**geom, 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 `stats::bw.nrd()`.

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

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

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

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

Aesthetics

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

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

Learn more about setting these aesthetics in vignette("ggplot2-specs")

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
ggplot(diamonds, aes(carat)) + geom_density()

ggplot(diamonds, aes(carat)) + geom_density(adjust = 1/5)

ggplot(diamonds, aes(carat)) + geom_density(adjust = 5)

ggplot(diamonds, aes(depth, colour = cut)) + geom_density() + xlim(55, 70)

ggplot(diamonds, aes(depth, fill = cut, colour = cut)) + geom_density(alpha = 0.1) + xlim(55, 70)
```

# 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

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

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

---

**geom_density_2d**  
*Contours of a 2d density estimate*

**Description**

Perform a 2D kernel density estimation using `MASS::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 = 10, 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).
geom_density_2d

- **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. It can also be a named logical vector to finely select the aesthetics to display.
- **inherit.aes**: If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders()`.
- **geom, 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 `MASS::bandwidth.nrd()`.

### Aesthetics

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

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

Learn more about setting these aesthetics in vignette("ggplot2-specs")

### Computed 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 = stat(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
# + 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 = stat(density)), contour = FALSE)
# Or points:
d + stat_density_2d(geom = "point", aes(size = stat(density)), n = 20, contour = FALSE)

dotsize = 1

---

**geom_dotplot**

*Dot plot*

**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 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.

**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.

**dotsize**
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. It can also be a named logical vector to finely select the aesthetics to display.

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

**Details**

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

Learn more about setting these aesthetics in vignette("ggplot2-specs")

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

ggplot(mtcars, aes(x = mpg)) + geom_dotplot()
ggplot(mtcars, aes(x = mpg)) + geom_dotplot(binwidth = 1.5)

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

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

# y axis isn't really meaningful, so hide it
ggplot(mtcars, aes(x = mpg)) + geom_dotplot(binwidth = 1.5) +
  scale_y_continuous(NULL, breaks = NULL)

# Overlap dots vertically
ggplot(mtcars, aes(x = mpg)) + geom_dotplot(binwidth = 1.5, stackratio = .7)
# Expand dot diameter
ggplot(mtcars, aes(x = mpg)) + geom_dotplot(binwidth = 1.5, dotsize = 1.25)

# Examples with stacking along y axis instead of x
ggplot(mtcars, aes(x = 1, y = mpg)) +
  geom_dotplot(binaxis = "y", stackdir = "center")

ggplot(mtcars, aes(x = factor(cyl), y = mpg)) +
  geom_dotplot(binaxis = "y", stackdir = "center")

ggplot(mtcars, aes(x = factor(cyl), y = mpg)) +
  geom_dotplot(binaxis = "y", stackdir = "centerwhole")

ggplot(mtcars, aes(x = factor(vs), fill = factor(cyl), y = mpg)) +
  geom_dotplot(binaxis = "y", stackdir = "center", position = "dodge")

# binpositions="all" ensures that the bins are aligned between groups

ggplot(mtcars, aes(x = factor(am), y = mpg)) +
  geom_dotplot(binaxis = "y", stackdir = "center", binpositions="all")

# Stacking multiple groups, with different fill

ggplot(mtcars, aes(x = mpg, fill = factor(cyl))) +
  geom_dotplot(stackgroups = TRUE, binwidth = 1, binpositions = "all")

ggplot(mtcars, aes(x = mpg, fill = factor(cyl))) +
  geom_dotplot(stackgroups = TRUE, binwidth = 1, method = "histodot")

ggplot(mtcars, aes(x = 1, y = mpg, fill = factor(cyl))) +
  geom_dotplot(binaxis = "y", stackgroups = TRUE, binwidth = 1, method = "histodot")

---

### geom_errorbarh

**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.
Geom Errorbarh

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. It can also be a named logical vector to finely select the aesthetics to display.

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

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

- `xmin`
- `xmax`
- `y`
- `alpha`
- `colour`
- `group`
- `height`
- `linetype`
- `size`

Learn more about setting these aesthetics in vignette("ggplot2-specs")

Examples

```r
df <- data.frame(
  trt = factor(c(1, 1, 2, 2)),
  resp = c(1, 5, 3, 4),
  group = factor(c(1, 2, 1, 2)),
)```
histogram

## 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 the levels of a categorical variable.

## Usage

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

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

```r
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. It can also be a named logical vector to finely select the aesthetics to display.

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

binwidth The width of the bins. Can be specified as a numeric value, or a function that calculates width from x. 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.

gem, 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. Overrides 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

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

ggplot(diamonds, aes(carat)) + geom_histogram(binwidth = 0.01)

ggplot(diamonds, aes(carat)) + geom_histogram(bins = 200)

# Rather than stacking histograms, it's easier to compare frequency polygons
ggplot(diamonds, aes(price, fill = cut)) + geom_histogram(binwidth = 500)

# 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, stat(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()

# You can specify a function for calculating binwidth,
# particularly useful when faceting along variables with
# different ranges
mtlong <- reshape2::melt(mtcars)
ggplot(mtlong, aes(value)) + facet_wrap(~variable, scales = 'free_x') +
  geom_histogram(binwidth = function(x) 2 * IQR(x) / (length(x)^1/3))

---

**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**

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

```r
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. It can also be a named logical vector to finely select the aesthetics to display.

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

**geom, 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**

Learn more about setting these aesthetics in vignette("ggplot2-specs")
See Also

`stat_bin2d()` for rectangular binning

Examples

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))

---

table
ggplot2

**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.
geom_jitter

- **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.
- **width** and **height**: 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. It can also be a named logical vector to finely select the aesthetics to display.
- **inherit.aes**: If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders()`.

**Aesthetics**

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

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

Learn more about setting these aesthetics in vignette("ggplot2-specs")
See Also

gem_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(cty, hwy)) + geom_jitter()

ggplot(mpg, aes(cty, hwy)) + geom_jitter(width = 0.5, height = 0.5)
```

---

**Description**

geom_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,
label.padding = unit(0.25, "lines"), label.r = unit(0.15, "lines"),
label.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,
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.

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. It
can also be a named logical vector to finely select the aesthetics to display.

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

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

Details
Note that 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 the plot is not constant in data units: when you resize a plot, labels stay the same
size, but the size of the axes changes.

geom_text and geom_label both add a label for each row in the data, even if coordinates x, y are
set to single values in the call to geom_label or geom_text. To add labels at specified points use
annotate() with annotate(geom = "text", ...) or annotate(geom = "label", ...).
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

Learn more about setting these aesthetics in vignette("ggplot2-specs")

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

```r
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:

```r
p + geom_text(family = "Times New Roman")
```

## Add aesthetic mappings

```r
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)

```r
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.

```r
p + geom_text(aes(label = paste(wt, "^cyl, ", sep = "")),
  parse = TRUE)
```

## Add a text annotation

```r
p +
  geom_text() +
  annotate("text", label = "plot mpg vs. wt", x = 2, y = 15, size = 8, colour = "red")
```

## Aligning labels and bars

```r
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:

```r
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:

```r
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

```r
ggplot(data = df, aes(x, y, group = grp)) +
  geom_col(aes(fill = grp), position = "dodge") +
  geom_text(
    aes(label = y, y = y + 0.05),
    position = position_dodge(0.9),
    vjust = 0
  )
```
To place text in the middle of each bar in a stacked barplot, you need to set the vjust parameter of position_stack().

```r
# ggplot(data = df, aes(x, y, group = grp)) + geom_col(aes(fill = grp)) + geom_text(aes(label = y), position = position_stack(vjust = 0.5))

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")
)

# Justification ---------------------------------------------------------------
ggplot(df, aes(x, y)) + geom_text(aes(label = text))
ggplot(df, aes(x, y)) + geom_text(aes(label = text), vjust = "inward", hjust = "inward")

---

text

**geom_map**

### Polygons from a reference map

#### Description

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

#### Usage

```r
gem_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.

- **...**
  - 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.
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`.

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

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

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

### 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.

```r
ids <- factor(c("1.1", "2.1", "1.2", "2.2", "1.3", "2.3"))

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

positions <- 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)
)

ggplot(values) +
geom_path(aes(map_id = id), map = positions) +
expand_limits(positions)
ggplot(values, aes(fill = value)) +
geom_map(aes(map_id = id), map = positions) +
expand_limits(positions)
ggplot(values, aes(fill = value)) +
geom_map(aes(map_id = id), map = positions) +
expand_limits(positions) + ylim(0, 3)

# Better example
crimes <- data.frame(state = tolower(rownames(USArrests)), USArrests)
crimesm <- reshape2::melt(crimes, id = 1)
if (require(maps)) {
states_map <- map_data("state")
ggplot(crimes, aes(map_id = state)) +
  geom_map(aes(fill = Murder), map = states_map) +
  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**

```r
geom_path(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", ..., lineend = "butt", linejoin = "round",
  linemitre = 10, 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, ...)
```
geom_path

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).

arrow  Arrow specification, as created by grid::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. It can also be a named logical vector to finely select the aesthetics to display.

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

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

Details

An alternative parameterisation is geom_segment(), where 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
geom_path

- alpha
- colour
- group
- linetype
- size

Learn more about setting these aesthetics in vignette("ggplot2-specs")

See Also

geom_polygon(): Filled paths (polygons); geom_segment(): Line segments

Examples

```r
# geom_line() is suitable for time series
ggplot(economics, aes(date, employ)) + 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, employ)) + geom_line()

# 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_line()

# Changing parameters

# 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
```
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

```r
ggplot(df) + geom_point() + geom_line()
```

### 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.
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.

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

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.

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

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

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

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.

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 = "."))

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

- x
- y
geom_point

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

Learn more about setting these aesthetics in vignette("ggplot2-specs")

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))

# 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) +
p + geom_point(colour = "pink", size = 4, show.legend = TRUE) +
```
Geom Polygons

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

```
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. It can also be a named logical vector to finely select the aesthetics to display.

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

Aesthetics

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

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

Learn more about setting these aesthetics in vignette("ggplot2-specs")

See Also

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

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

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

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

  positions <- 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, 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
geom_qq_line

Description

geom_qq and stat_qq produce quantile-quantile plots. geom_qq_line and stat_qq_line compute the slope and intercept of the line connecting the points at specified quartiles of the theoretical and sample distributions.

Usage

geom_qq_line(mapping = NULL, data = NULL, geom = "path", position = "identity", ..., distribution = stats::qnorm, dparams = list(), line.p = c(0.25, 0.75), fullrange = FALSE, na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)

stat_qq_line(mapping = NULL, data = NULL, geom = "path", position = "identity", ..., distribution = stats::qnorm, dparams = list(), line.p = c(0.25, 0.75), fullrange = FALSE, na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)

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

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

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

gem  
The geometric object to use display the data

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

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

distribution  
Distribution function to use, if \texttt{x} not specified

dparams  
Additional parameters passed on to \texttt{distribution} function.

line.p  
Vector of quantiles to use when fitting the Q-Q line, defaults defaults to \texttt{c(.25, .75)}.

fullrange  
Should the q-q line span the full range of the plot, or just the data

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

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

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

Aesthetics

\texttt{stat_qq} understands the following aesthetics (required aesthetics are in bold):

- sample
- group
- x
- y

Learn more about setting these aesthetics in \texttt{vignette("ggplot2-specs")}

\texttt{stat_qq_line} understands the following aesthetics (required aesthetics are in bold):

- sample
• group
• x
• y

Learn more about setting these aesthetics in vignette("ggplot2-specs")

Computed variables

Variables computed by stat_qq:

**sample**  sample quantiles
**theoretical**  theoretical quantiles

Variables computed by stat_qq_line:

**x**  x-coordinates of the endpoints of the line segment connecting the points at the chosen quantiles of the theoretical and the sample distributions
**y**  y-coordinates of the endpoints

Examples

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

# 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"])[1] +
  stat_qq_line(distribution = qt, dparams = params["df"])[1]

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

---

**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

gem_quantile(mapping = NULL, data = NULL, stat = "quantile",
    position = "identity", ..., lineend = "butt", linejoin = "round",
    linemitre = 10, na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)

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. It can also be a named logical vector to finely select the aesthetics to display.

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

geom, 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 Quatile regression method to use. Currently only supports quantreg::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

Learn more about setting these aesthetics in vignette("ggplot2-specs")

Computed variables

quantile quantile of distribution

Examples

m <- ggplot(mpg, aes(displ, 1 / hwy)) + geom_point()
m + geom_quantile()
m + geom_quantile(quantiles = 0.5)
q10 <- seq(0.05, 0.95, by = 0.05)
m + geom_quantile(quantiles = q10)

# 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
# you'll need to tweak lambda yourself
m + geom_quantile(method = "rqss", lambda = 0.1)

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

---

**geom_raster**

**Rectangles**

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.
geom_raster

Usage

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)

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

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 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.

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. It
can also be a named logical vector to finely select the aesthetics to display.

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

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

- x
- y
- alpha
- colour
- fill
- group
- height
- linetype
- size
- width

Learn more about setting these aesthetics in vignette("ggplot2-specs")

Examples

# 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), colour = "grey50")

ggplot(df, aes(x, y, width = w)) +
  geom_tile(aes(fill = z), colour = "grey50")

ggplot(df, aes(xmin = x - w / 2, xmax = x + w / 2, ymin = y, ymax = y + 1)) +
  geom_rect(aes(fill = z), 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()`

# zero padding
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

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

```r
ggplot(df, aes(x, y, fill = z)) + geom_raster(hjust = 0, vjust = 0)
```

```r
# Inspired by the image-density plots of Ken Knoblauch
cars <- ggplot(mtcars, aes(mpg, factor(cyl)))
cars + geom_point()
cars + stat_bin2d(aes(fill = stat(count)), binwidth = c(3,1))
cars + stat_bin2d(aes(fill = stat(density)), binwidth = c(3,1))
cars + stat_density(aes(fill = stat(density)), geom = "raster", position = "identity")
cars + stat_density(aes(fill = stat(count)), geom = "raster", position = "identity")
```

### 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. It can also be a named logical vector to finely select the aesthetics to display.
- `inherit.aes` If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders()`.

Details

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`

Learn more about setting these aesthetics in vignette("ggplot2-specs")

See Also

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

Examples

```r
# 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))
```
geom_rug

```r
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))
```

---

### 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

```r
gem_rug(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", ..., sides = "btl", 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. It can also be a named logical vector to finely select the aesthetics to display.

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

Details

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

Aesthetics

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

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

Learn more about setting these aesthetics in vignette("ggplot2-specs")

Examples

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

# Use jittering to avoid overplotting for smaller datasets
ggplot(mpg, aes(displ, cty)) +
  geom_point() +
  geom_rug()

ggplot(mpg, aes(displ, cty)) +
  geom_jitter() +
  geom_rug(alpha = 1/2, position = "jitter")
```
geom_segment  Line segments and curves

Description

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

Usage

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

gem_curve(mapping = NULL, data = NULL, stat = "identity",
position = "identity", ..., curvature = 0.5, angle = 90, ncp = 5,
arrow = NULL, arrow.fill = 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()`.

arrow.fill fill color to use for the arrow head (if closed). NULL means use colour aesthetic.

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

linejoin Line join style (round, mitre, bevel).
na.rm If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

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

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

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

d4m_segment understands the following aesthetics (required aesthetics are in bold):

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

Learn more about setting these aesthetics in vignette("ggplot2-specs")

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

```r
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_segment(aes(x = x1, y = y1, xend = x2, yend = y2, colour = "segment"), data = df)

b + geom_curve(aes(x = x1, y = y1, xend = x2, yend = y2), data = df, curvature = -0.2)
b + geom_curve(aes(x = x1, y = y1, xend = x2, yend = y2), data = df, curvature = 1)
b + geom_curve(aes(x = x1, y = y1, xend = x2, yend = y2),
data = df,
  arrow = arrow(length = unit(0.03, "npc")))

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")

# Use lineend and linejoin to change the style of the segments
df2 <- expand.grid(
  lineend = c('round', 'butt', 'square'),
  linejoin = c('round', 'mitre', 'bevel'),
  stringsAsFactors = FALSE)
df2 <- data.frame(df2, y = 1:9)
ggplot(df2, aes(x = 1, y = y, xend = 2, yend = y, label = paste(lineend, linejoin))) +
  geom_segment(
    lineend = df2$lineend, linejoin = df2$linejoin,
    size = 3, arrow = arrow(length = unit(0.3, "inches")))
  ) +
  geom_text(hjust = 'outside', nudge_x = -0.2) +
xlim(0.5, 2)

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

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

**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
geom_smooth(mapping = NULL, data = NULL, stat = "smooth",
position = "identity", ..., method = "auto", formula = y ~ x,
se = TRUE, na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)

stat_smooth(mapping = NULL, data = NULL, geom = "smooth",
position = "identity", ..., method = "auto", formula = y ~ x,
se = TRUE, n = 80, span = 0.75, fullrange = FALSE, level = 0.95,
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.

- **method**: Smoothing method (function) to use, eg. `lm`, `glm`, `loess`, `MASS::rlm`.
  - For `method = "auto"` the smoothing method is chosen based on the size of the largest group (across all panels). `loess()` is used for less than 1,000 observations; otherwise `mgcv::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.
  - If you have fewer than 1,000 observations but want to use the same `gam` model that `method = "auto"` would use then set `method = "gam", formula = y ~ s(x, bs = "cs")`.

- **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 control.)
geom_smooth

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

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

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

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

n Number of points at which to evaluate smoother.

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 method.

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
- ymax
- ymin

Learn more about setting these aesthetics in vignette("ggplot2-specs")
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:
# small numbers make a wigglier curve, larger numbers make a smoother curve.
intersection(ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_smooth(span = 0.3)

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

intersection(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
intersection(ggplot(mpg, aes(displ, hwy, colour = class)) +
  geom_point() +
  geom_smooth(se = FALSE, method = lm)

intersection(ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_smooth(span = 0.8) +
  facet_wrap(~drv)

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.
intersection(ggplot(rpart::kyphosis, aes(Age, Kyphosis)) +
  geom_point() +
  geom_smooth(binomial_smooth, method.args = list(family = "binomial"), se = FALSE) +
  geom_point()

```

```
geom_jitter(height = 0.05) +
binomial_smooth()

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

# 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

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapping</td>
<td>Set of aesthetic mappings created by <code>aes()</code> or <code>aes_()</code>. If specified and <code>inherit.aes = TRUE</code> (the default), it is combined with the default mapping at the top level of the plot. You must supply <code>mapping</code> if there is no plot mapping.</td>
</tr>
<tr>
<td>data</td>
<td>The data to be displayed in this layer. There are three options: If <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code>. A <code>data.frame</code>, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data.</td>
</tr>
<tr>
<td>stat</td>
<td>The statistical transformation to use on the data for this layer, as a string.</td>
</tr>
<tr>
<td>position</td>
<td>Position adjustment, either as a string, or the result of a call to a position adjustment function.</td>
</tr>
</tbody>
</table>
... 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. It can also be a named logical vector to finely select the aesthetics to display.

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

**Aesthetics**

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

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

Learn more about setting these aesthetics in vignette("ggplot2-specs")

**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))
```
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
gem_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)
```

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

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

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

Use to override the default connection between `geom_violin` and `stat_ydensity`.

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

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

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

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

Learn more about setting these aesthetics in `vignette("ggplot2-specs")`

**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
geom_violin

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
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**

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
  DEPRECATED. Used prior to tidy evaluation.

**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:

- ggplot(df, aes(x, y, other aesthetics))
- ggplot(df)
- 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

```r
# 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
# mult-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
  )
```

---

**ggproto**

Create a new ggproto object

**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  
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

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**

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** Multiplicative scaling factor.
- **width, height, units** Plot size in units ("in", "cm", or "mm"). If not supplied, uses the size of current graphics device.
- **dpi** Plot resolution. Also accepts a string input: "retina" (320), "print" (300), or "screen" (72). 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)
```

---

**ggsf**

*Visualise sf objects*

**Description**

This set of geom, stat, and coord are used to visualise simple feature (sf) objects. For simple plots, you will only need `geom_sf` as it uses `stat_sf` and adds `coord_sf` for you. `geom_sf` is an unusual geom because it will draw different geometric objects depending on what simple features are present in the data: you can get points, lines, or polygons.

**Usage**

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

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

coord_sf(xlim = NULL, ylim = NULL, expand = TRUE, crs = NULL,
datum = sf::st_crs(4326), ndiscr = 100, default = FALSE)
```

**Arguments**

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

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.

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

logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
You can also set this to one of "polygon", "line", and "point" to override the default legend.

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

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

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

Limits for the x and y axes.

Limits for the x and y axes.

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`.

Use this to select a specific CRS. If not specified, will use the CRS defined in the first layer.

CRS that provides datum to use when generating graticules

number of segments to use for discretizing graticule lines; try increasing this when graticules look unexpected

Is this the default coordinate system? If FALSE (the default), then replacing this coordinate system with another one creates a message alerting the user that the coordinate system is being replaced. If TRUE, that warning is suppressed.

Geometry aesthetic

`geom_sf` uses a unique aesthetic: `geometry`, giving an column of class `sfc` containing simple features data. There are three ways to supply the geometry aesthetic:

- Do nothing: by default `geom_sf` assumes it is stored in the geometry column.
• Explicitly pass an sf object to the data argument. This will use the primary geometry column, no matter what it’s called.
• Supply your own using aes(geometry = my_column)

Unlike other aesthetics, geometry will never be inherited from the plot.

CRS

coord_sf() ensures that all layers use a common CRS. You can either specify it using the CRS param, or coord_sf will take it from the first layer that defines a CRS.

Examples

```r
if (requireNamespace("sf", quietly = TRUE)) {
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
  ggplot(nc) +
    geom_sf(aes(fill = AREA))

  # If not supplied, coord_sf() will take the CRS from the first layer
  # and automatically transform all other layers to use that CRS. This
  # ensures that all data will correctly line up
  nc_3857 <- sf::st_transform(nc, "+init=epsg:3857")
  ggplot() +
    geom_sf(data = nc) +
    geom_sf(data = nc_3857, colour = "red", fill = NA)

  # Unfortunately if you plot other types of feature you'll need to use
  # show.legend to tell ggplot2 what type of legend to use
  nc_3857$mid <- sf::st_centroid(nc_3857$geometry)
  ggplot(nc_3857) +
    geom_sf(colour = "white") +
    geom_sf(aes(geometry = mid, size = AREA), show.legend = "point")

  # You can also use layers with x and y aesthetics: these are
  # assumed to already be in the common CRS.
  ggplot(nc) +
    geom_sf() +
    annotate("point", x = -80, y = 35, colour = "red", size = 4)

  # Thanks to the power of sf, a geom_sf nicely handles varying projections
  # setting the aspect ratio correctly.
  library(maps)
  world1 <- sf::st_as_sf(map('world', plot = FALSE, fill = TRUE))
  ggplot() + geom_sf(data = world1)
  world2 <- sf::st_transform( world1,
    "+proj=laea +y_0=0 +lon_0=155 +lat_0=-90 +ellps=WGS84 +no_defs"
  )
  ggplot() + geom_sf(data = world2)
}
```
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

```r
theme_grey(base_size = 11, base_family = "", base_line_size = base_size/22, base_rect_size = base_size/22)
theme_gray(base_size = 11, base_family = "", base_line_size = base_size/22, base_rect_size = base_size/22)
theme_bw(base_size = 11, base_family = "", base_line_size = base_size/22, base_rect_size = base_size/22)
theme_linedraw(base_size = 11, base_family = "", base_line_size = base_size/22, base_rect_size = base_size/22)
theme_light(base_size = 11, base_family = "", base_line_size = base_size/22, base_rect_size = base_size/22)
theme_dark(base_size = 11, base_family = "", base_line_size = base_size/22, base_rect_size = base_size/22)
theme_minimal(base_size = 11, base_family = "", base_line_size = base_size/22, base_rect_size = base_size/22)
theme_classic(base_size = 11, base_family = "", base_line_size = base_size/22, base_rect_size = base_size/22)
theme_void(base_size = 11, base_family = "", base_line_size = base_size/22, base_rect_size = base_size/22)
theme_test(base_size = 11, base_family = "", base_line_size = base_size/22, base_rect_size = base_size/22)
```

Arguments

- `base_size` base font size
- `base_family` base font family
- `base_line_size` base size for line elements
- `base_rect_size` base size for rect elements
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.
theme_test  A theme for visual unit tests. It should ideally never change except for new features.

Examples

```r
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

```
# 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
p + guides(colour = "none")
p + guides(colour = "colorbar", size = "none")

# Guides are integrated where possible

p + guides(colour = guide_legend("title"), size = guide_legend("title"),
           shape = guide_legend("title"))
# same as
g <- guide_legend("title")
p + guides(colour = g, size = g, shape = g)

p + theme(legend.position = "bottom")

# position of guides

# Set order for multiple guides
ggplot(mpg, aes(displ, cty)) +
       geom_point(aes(size = hwy, colour = cyl, shape = drv)) +
       guides(
                  colour = guide_colourbar(order = 1),
                  shape = guide_legend(order = 2),
```
size = guide_legend(order = 3)
)


---

guide_colourbar

---

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, frame.colour = NULL,
frame.linewidth = 0.5, frame.linetype = 1, ticks = TRUE,
ticks.colour = "white", ticks.linewidth = 0.5, draw.ullim = TRUE,
draw.llim = TRUE, direction = NULL, default.unit = "line",
reverse = FALSE, order = 0, available_aes = c("colour", "color",
"fill"), ...)

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, frame.colour = NULL,
frame.linewidth = 0.5, frame.linetype = 1, ticks = TRUE,
ticks.colour = "white", ticks.linewidth = 0.5, draw.ullim = TRUE,
draw.llim = TRUE, direction = NULL, default.unit = "line",
reverse = FALSE, order = 0, available_aes = c("colour", "color",
"fill"), ...)

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.
```r
guide_colourbar

- `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 `grid::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 `grid::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.
- `frame.colour`: A string specifying the colour of the frame drawn around the bar. If `NULL` (the default), no frame is drawn.
- `frame.linewidth`: A numeric specifying the width of the frame drawn around the bar.
- `frame.linetype`: A numeric specifying the linetype of the frame drawn around the bar.
- `ticks`: A logical specifying if tick marks on colorbar should be visible.
- `ticks.colour`: A string specifying the color of the tick marks.
- `ticks.linewidth`: A numeric specifying the width of the tick marks.
- `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 `grid::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 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.
- `available_aes`: A vector of character strings listing the aesthetics for which a colorbar can be drawn.
- `...`: 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 `grid::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 `grid::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 `grid::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.
...          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_colourbar`, `guides`

Examples

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

pl <- ggplot(def, aes(X1, X2)) + geom_tile(aes(fill = value))
pl2 <- pl1 + geom_point(aes(size = value))

# Basic form
pl + scale_fill_continuous(guide = guide_legend())
# Control styles

# 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(mtcars, aes(vs, am, colour = factor(cyl))) +
  geom_jitter(alpha = 1/5, width = 0.01, height = 0.01)
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])
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))

# reversed order legend
p + guides(col = guide_legend(reverse = TRUE))
hmisc

A selection of summary functions from Hmisc

Description
These are wrappers around functions from **Hmisc** designed to make them easier to use with **stat_summary()**. See the Hmisc documentation for more details:

- `Hmisc::smean.cl.boot()`
- `Hmisc::smean.cl.normal()`
- `Hmisc::smean.sdl()`
- `Hmisc::smedian.hilow()`

Usage

```r
mean_cl_boot(x, ...)
mean_cl_normal(x, ...)
mean_sdl(x, ...)
median_hilow(x, ...)
```

Arguments

- `x` a numeric vector
- `...` other arguments passed on to the respective Hmisc function.

Value

A data frame with columns `y`, `ymin`, and `ymax`.

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

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 Hmisc::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$conervation2 <- plyr::revalue(msleep$conervation, conservation_status)
p3 <- ggplot(msleep, aes(x = sleep_total, y = awake)) + geom_point()
p3 + facet_grid(vore ~ conservation2,
  labeller = labeller(conservation2 = label_wrap_gen(10))
)

# labeller() is especially useful to act as a global labeller. You
# 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)
p3 + facet_wrap(~conservation2, labeller = global_labeller)

---

**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 faceting 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 faceting and label_both() when multiple factors are involved. label_wrap_gen() uses base::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()

# The default is label_value
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
# depending on whether multiple factors are facetted over
```
### 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

```r
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

```r
# 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(am ~ 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)))
```
**labs**  
*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. *tag* can be used for adding identification tags.

**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", "caption", or "tag".
- `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  
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  
p + labs(caption = "(based on data from ...)")
```
The plot tag appears at the top-left, and is typically used for labelling a subplot with a letter.

```r
p + labs(title = "title", tag = "A")
```

### lims

**Set scale limits**

#### 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:

<table>
<thead>
<tr>
<th>L, u, v</th>
<th>Position in Luv colour space</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>Colour name</td>
</tr>
</tbody>
</table>

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

```r
caption = NULL, caption.theme = NULL, caption.position = NULL, desc = NULL, desc.theme = NULL, desc.position = NULL, debug = FALSE, inherit = FALSE)
```

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 Line end style (round, butt, square)
- `arrow` Arrow specification, as created by `grid::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.
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
  )
)

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
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.
- **perblack** Percent black.
- **percamerindian** Percent American Indian.
- **percasian** Percent Asian.
- **percother** Percent other races.
- **popadults** Number of adults.
- **perchsd**
- **percollege** Percent college educated.
percprof  Percent profession.
poppovertyknown
percpovertyknown
percbelowpoverty
percchildbelowpovert
percadultpoverty
percelderypoverty
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. 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
city  city miles per gallon
hwy   highway miles per gallon
fl     fuel type
class  "type" of car
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

tvore  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. position_dodge is a special case of position_dodge for arranging box plots, which can have variable widths. position_dodge also works with bars and rectangles.

Usage

position_dodge(width = NULL, preserve = c("total", "single"))

position_dodge2(width = NULL, preserve = c("total", "single"),
    padding = 0.1, reverse = FALSE)

Arguments

width          Dodging width, when different to the width of the individual elements. This is useful when you want to align narrow geoms with wider geoms. See the examples.

preserve       Should dodging preserve the total width of all elements at a position, or the width of a single element?

padding        Padding between elements at the same position. Elements are shrunk by this proportion to allow space between them. Defaults to 0.1.

reverse        If TRUE, will reverse the default stacking order. This is useful if you’re rotating both the plot and legend.

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 = "dodge2")

# By default, dodging with `position_dodge2()` preserves the width of each element. You can choose to preserve the total width with:
ggplot(mtcars, aes(factor(cyl), fill = factor(vs))) +
geom_bar(position = position_dodge(preserve = "total"))

ggplot(diamonds, aes(price, fill = cut)) +
geom_histogram(position="dodge2")
# see ?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 = position_dodge(width = 0.9)
)

# The same principle applies to error bars, which are usually
# narrower than the bars
p + geom_errorbar(
  aes(ymin = y - 1, ymax = y + 1),
  width = 0.2,
  position = "dodge"
)
p + geom_errorbar(
  aes(ymin = y - 1, ymax = y + 1),
  width = 0.2,
  position = position_dodge(width = 0.9)
)

# Box plots use position_dodge2 by default, and bars can use it too
ggplot(data = iris, aes(Species, Sepal.Length)) +
  geom_boxplot(aes(colour = Sepal.Width < 3.2))

ggplot(data = iris, aes(Species, Sepal.Length)) +
  geom_boxplot(aes(colour = Sepal.Width < 3.2), varwidth = TRUE)

ggplot(mtcars, aes(factor(cyl), fill = factor(vs))) +
  geom_bar(position = position_dodge2(preserve = "single"))

ggplot(mtcars, aes(factor(cyl), fill = factor(vs))) +
  geom_bar(position = position_dodge2(preserve = "total"))
```

Don’t adjust position
position_jitter

**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, seed = NA)

**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.

- **seed** A random seed to make the jitter reproducible. Useful if you need to apply the same jitter twice, e.g., for a point and a corresponding label. The random seed is reset after jittering. If NA (the default value), the seed is initialised with a random value; this makes sure that two subsequent calls start with a different seed. Use NULL to use the current random seed and also avoid resetting (the behavior of ggplot 2.2.1 and earlier).

**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)
ggplot(mtcars, aes(am, vs)) +
  geom_jitter(position = position_jitter(width = 0.1, height = 0.1))

# Create a jitter object for reproducible jitter:
jitter <- position_jitter(width = 0.1, height = 0.1)
ggplot(mtcars, aes(am, vs)) +
  geom_point(position = jitter) +
  geom_point(position = jitter, color = "red", aes(am + 0.2, vs + 0.2))
```

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, seed = NA)
```

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.
- `seed`: A random seed to make the jitter reproducible. Useful if you need to apply the same jitter twice, e.g., for a point and a corresponding label. The random seed is reset after jittering. If NA (the default value), the seed is initialised with a random value; this makes sure that two subsequent calls start with a different seed. Use NULL to use the current random seed and also avoid resetting (the behavior of `ggplot` 2.2.1 and earlier).
position_nudge

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**

- **ggplot**
  ```r
ggplot(df, aes(x, y)) +
  geom_point() +
  geom_text(aes(label = y), nudge_y = -0.1)
  ```

---

**position_stack**

*Stack overlapping objects on top of each other*

**Description**

*position_stack*() stacks bars on top of each other; *position_fill*() stacks bars and standardises each stack to have constant height.

**Usage**

```r
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()

# Stacking is also useful for time series
series <- data.frame(
  time = c(rep(1, 4), rep(2, 4), rep(3, 4), rep(4, 4)),
  type = rep(c('a', 'b', 'c', 'd'), 4),
  value = rpois(16, 10))

# The stacking order is carefully designed so that the plot matches
# the legend.

# You control the stacking order by setting the levels of the underlying
# factor. See the forcats package for convenient helpers.
series$type2 <- factor(series$type, levels = c('c', 'b', 'd', 'a'))

# You can change the order of the levels in the legend using the scale

# If you've flipped the plot, use reverse = TRUE so the levels
# continue to match

# 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.

presidential

```
geom_line(aes(colour = type), position = "stack") +
geom_point(aes(colour = type), position = "stack")

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.
  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
  geom_text(aes(label = type), position = position_stack(vjust = 0.5))

# Negative values  

df <- tibble::tribble(~x, ~y, ~grp, 
  "a", 1, "x", 
  "a", 2, "y", 
  "b", 1, "x", 
  "b", 3, "y", 
  "b", -1, "y" )

ggplot(data = df, aes(x, y, group = grp)) +
  geom_col(aes(fill = grp), position = position_stack(reverse = TRUE)) +
  geom_hline(yintercept = 0)

ggplot(data = df, 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.

**Usage**

`presidential`
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
```r
## 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
```r
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")
  } )
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

qplot(x, y, ..., data, facets = NULL, margins = FALSE, geom = "auto",
    xlim = c(NA, NA), ylim = c(NA, NA), log = "", main = NULL,
    xlab = NULL, ylab = NULL, asp = NA, stat = NULL, position = NULL)

quickplot(x, y, ..., data, facets = NULL, margins = FALSE, geom = "auto",
    xlim = c(NA, NA), ylim = c(NA, NA), log = "", main = NULL,
    xlab = NULL, ylab = NULL, 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**

*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)`

**Arguments**

- `x` numeric vector
- `zero` should a zero value be automatically included in the computation of resolution

**Examples**

```r
resolution(1:10)
resolution((1:10) - 0.5)
resolution((1:10) - 0.5, FALSE)
```

# Note the difference between numeric and integer vectors
```r
resolution(c(2, 10, 20, 50))
resolution(c(2L, 10L, 20L, 50L))
```
scale_alpha

*Alpha transparency scales*

**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**

```r
scale_alpha(..., range = c(0.1, 1))
scale_alpha_continuous(..., range = c(0.1, 1))
scale_alpha_discrete(...)
scale_alpha_ordinal(..., 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`, `scale_colour_viridis_d`

**Examples**

```r
p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(alpha = year))

p
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](http://colorbrewer2.org) for more information.

**Usage**

```r
scale_colour_brewer(..., type = "seq", palette = 1, direction = 1,
                  aesthetics = "colour")

scale_fill_brewer(..., type = "seq", palette = 1, direction = 1,
                  aesthetics = "fill")

scale_colour_distiller(..., type = "seq", palette = 1, direction = -1,
                       values = NULL, space = "Lab", na.value = "grey50",
                       guide = "colourbar", aesthetics = "colour")

scale_fill_distiller(..., type = "seq", palette = 1, direction = -1,
                       values = NULL, space = "Lab", na.value = "grey50",
                       guide = "colourbar", aesthetics = "fill")
```

**Arguments**

- `...`: Other arguments passed on to `discrete_scale()` or, for distiller scales, `continuous_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 `RColorBrewer::brewer_pal()`. If -1, the order of colors is reversed.
- `aesthetics`: Character string or vector of character strings listing the name(s) of the aesthetic(s) that this scale works with. This can be useful, for example, to apply colour settings to the colour and fill aesthetics at the same time, via aesthetics = c("colour", "fill").
- `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.
scale_colour_brewer

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.

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, YlOrBr, YlOrRd

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, scale_colour_viridis_d

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
scale_colour_continuous

Continuous colour scales

Description

Colour scales for continuous data default to the values of the `ggplot2.continuous.colour` and `ggplot2.continuous.fill` options. If these options are not present, "gradient" will be used. See `options()` for more information.

Usage

```r
scale_colour_continuous(...) type = getOption("ggplot2.continuous.colour",
  default = "gradient")
```

```r
scale_fill_continuous(...) type = getOption("ggplot2.continuous.fill", default
  = "gradient")
```

Arguments

... Additional parameters passed on to the scale type
type One of "gradient" (the default) or "viridis" indicating the colour scale to use

See Also

`scaleColourGradient()`, `scaleColourViridisC()`, `scaleFillGradient()`, and `scaleFillViridisC()`

Examples

```r
v <- ggplot(faithful, aes(waiting, eruptions, fill = density)) +
  geom_tile()

v
```

```r
v + scale_fill_continuous(type = "gradient")
v + scale_fill_continuous(type = "viridis")
```

# The above are equivalent to
```r
v + scale_fill_gradient()
v + scale_fill_viridis_c()
```
scale_colour_gradient

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\", aesthetics = \"colour\")

scale_fill_gradient(..., low = \"#132B43\", high = \"#56B1F7\", space = \"Lab\", na.value = \"grey50\", guide = \"colourbar\", aesthetics = \"fill\")

scale_colour_gradient2(..., low = muted("red"), mid = "white", high = muted("blue"), midpoint = 0, space = \"Lab\", na.value = \"grey50\", guide = \"colourbar\", aesthetics = \"colour\")

scale_fill_gradient2(..., low = muted("red"), mid = "white", high = muted("blue"), midpoint = 0, space = \"Lab\", na.value = \"grey50\", guide = \"colourbar\", aesthetics = \"fill\")

scale_colour_gradientn(..., colours, values = NULL, space = \"Lab\", na.value = \"grey50\", guide = \"colourbar\", aesthetics = \"colour\", colors)

scale_fill_gradientn(..., colours, values = NULL, space = \"Lab\", na.value = \"grey50\", guide = \"colourbar\", aesthetics = \"fill\", colors)

Arguments

... Arguments passed on to continuous_scale

aesthetics The names of the aesthetics that this scale works with

scale_name The name of the scale

palette A palette function that when called with a single integer argument (the number of levels in the scale) returns the values that they should take

name The name of the scale. Used as axis or legend title. If waiver(), the default, the name of the scale is taken from the first mapping used for that aesthetic. If NULL, the legend title will be omitted.

breaks One of:

- NULL for no breaks
- waiver() for the default breaks computed by the transformation object
- A numeric vector of positions
A function that takes the limits as input and returns breaks as output

**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.

**rescaler** Used by diverging and n colour gradients (i.e. `scale_colour_gradient2()`, `scale_colour_gradientn()`). A function used to scale the input values to the range $[0, 1]$.

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

**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, it’s inverse, and methods for generating breaks and labels. Transformation objects are defined in the scales package, and are called name_trans, e.g. `scales::boxcox_trans()`. You can create your own transformation with `scales::trans_new()`.

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

**super** The super class to use for the constructed scale

**expand** Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. Use the convenience function `expand_scale()` to generate the values for the expand argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

**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.

**aesthetics** Character string or vector of character strings listing the name(s) of the aesthetic(s) that this scale works with. This can be useful, for example, to apply colour settings to the colour and fill aesthetics at the same time, via aesthetics = c("colour", "fill").
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

scales::seq_gradient_pal() for details on underlying palette

Other colour scales: scale_alpha, scale_colour_brewer, scale_colour_grey, scale_colour_hue, scale_colour_viridis_d

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
ghplot(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
ghplot(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**  *Sequential grey colour scales*

**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",
  aesthetics = "colour")
```

```r
scale_fill_grey(..., start = 0.2, end = 0.8, na.value = "red",
  aesthetics = "fill")
```

**Arguments**

- `...` Arguments passed on to `discrete_scale`
- `breaks` One of:
  - `NULL` for no breaks
  - `waiver()` for the default breaks computed by the transformation object
  - A character vector of breaks
  - A function that takes the limits as input and returns breaks as output
- `limits` A character vector that defines possible values of the scale and their order.
- `drop` Should unused factor levels be omitted from the scale? The default, `TRUE`, uses the levels that appear in the data; `FALSE` uses all the levels in the factor.
- `na.translate` Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify `na.translate = FALSE`.
- `na.value` If `na.translate = TRUE`, what value aesthetic value should missing be displayed as? Does not apply to position scales where `NA` is always placed at the far right.
- `aesthetics` The names of the aesthetics that this scale works with
- `scale_name` The name of the scale
- `palette` A palette function that when called with a single integer argument (the number of levels in the scale) returns the values that they should take
- `name` The name of the scale. Used as axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.
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

expand Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. Use the convenience function expand_scale() to generate the values for the expand argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

guide A function used to create a guide or its name. See guides() for more info.

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

super The super class to use for the constructed scale

start gray value at low end of palette
end gray value at high end of palette
na.value Colour to use for missing values
aesthetics Character string or vector of character strings listing the name(s) of the aesthetic(s) that this scale works with. This can be useful, for example, to apply colour settings to the colour and fill aesthetics at the same time, via aesthetics = c("colour", "fill").

See Also

Other colour scales: scale_alpha, scale_colour_brewer, scale_colour_gradient, scale_colour_hue, scale_colour_viridis_d

Examples

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

scale_colour_hue(...) h = c(0, 360) + 15, c = 100, l = 65, h.start = 0,
  direction = 1, na.value = "grey50", aesthetics = "colour")

scale_fill_hue(...) h = c(0, 360) + 15, c = 100, l = 65, h.start = 0,
  direction = 1, na.value = "grey50", aesthetics = "fill")

Arguments

... Arguments passed on to discrete_scale

breaks One of:
  • NULL for no breaks
  • waiver() for the default breaks computed by the transformation object
  • A character vector of breaks
  • A function that takes the limits as input and returns breaks as output

limits A character vector that defines possible values of the scale and their order.

drop Should unused factor levels be omitted from the scale? The default, TRUE, uses the levels that appear in the data; FALSE uses all the levels in the factor.

na.translate Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify na.translate = FALSE.

na.value If na.translate = TRUE, what value aesthetic value should missing be displayed as? Does not apply to position scales where NA is always placed at the far right.

scale_name The name of the scale

palette A palette function that when called with a single integer argument (the number of levels in the scale) returns the values that they should take

name The name of the scale. Used as axis or legend title. If waiver(), the default, the name of the scale is taken from the first mapping used for that aesthetic. If NULL, the legend title will be omitted.

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

**expand**  Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. Use the convenience function `expand_scale()` to generate the values for the expand argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

**guide**  A function used to create a guide or its name. See `guides()` for more info.

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

**super**  The super class to use for the constructed scale

```
range of hues to use, in [0, 360]
```

```
chroma (intensity of colour), maximum value varies depending on combination of hue and luminance.
```

```
luminance (lightness), in [0, 100]
```

```
hue to start at
direction
direction to travel around the colour wheel, 1 = clockwise, -1 = counter-clockwise
```

```
Colour to use for missing values
```

```
Character string or vector of character strings listing the name(s) of the aesthetic(s) that this scale works with. This can be useful, for example, to apply colour settings to the colour and fill aesthetics at the same time, via aesthetics = c("colour", "fill").
```

See Also

Other colour scales: `scale_alpha`, `scale_colour_brewer`, `scale_colour_gradient`, `scale_colour-grey`, `scale_colour_viridis_d`

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)
d + scale_colour_hue(l = 70, c = 150)
d + scale_colour_hue(l = 80, c = 150)

# Change range of hues used
d + scale_colour_hue(h = c(0, 90))
```
scale_colour_viridis_d

Viridis colour scales from viridisLite

Description

The viridis scales provide color maps that are perceptually uniform in both color and black-and-white. They are also designed to be perceived by viewers with common forms of color blindness. See also https://bids.github.io/colormap/.

Usage

scale_colour_viridis_d(..., alpha = 1, begin = 0, end = 1,
  direction = 1, option = "D", aesthetics = "colour")

scale_fill_viridis_d(..., alpha = 1, begin = 0, end = 1, direction = 1,
  option = "D", aesthetics = "fill")

scale_colour_viridis_c(..., alpha = 1, begin = 0, end = 1,
  direction = 1, option = "D", values = NULL, space = "Lab",
  na.value = "grey50", guide = "colourbar", aesthetics = "colour")

scale_fill_viridis_c(..., alpha = 1, begin = 0, end = 1, direction = 1,
  option = "D", values = NULL, space = "Lab", na.value = "grey50",
  guide = "colourbar", aesthetics = "fill")

Arguments

... Other arguments passed on to discrete_scale() or continuous_scale() to control name, limits, breaks, labels and so forth.
alpha The alpha transparency, a number in [0,1], see argument alpha in hsv.
begin The (corrected) hue in [0,1] at which the viridis colormap begins.
end The (corrected) hue in [0,1] at which the viridis colormap ends.
direction Sets the order of colors in the scale. If 1, the default, colors are ordered from
darkest to lightest. If -1, the order of colors is reversed.
option A character string indicating the colormap option to use. Four options are avail-
able: "magma" (or "A"), "inferno" (or "B"), "plasma" (or "C"), "viridis" (or "D", the
default option) and "cividis" (or "E").
aesthetics Character string or vector of character strings listing the name(s) of the aes-
thetic(s) that this scale works with. This can be useful, for example, to apply
colour settings to the colour and fill aesthetics at the same time, via
aesthetics = c("colour", "fill").
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 convenience 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 Missing values will be replaced with this value.
guide A function used to create a guide or its name. See guides() for more info.

See Also
Other colour scales: scale_alpha, scale_colour_brewer, scale_colour_gradient, scale_colour_grey,
scale_colour_hue

Examples

# viridis is the default colour/fill scale for ordered factors
dsamp <- diamonds[sample(nrow(diamonds), 1000),]
ggplot(dsamp, aes(carat, price)) +
  geom_point(aes(colour = clarity))

# Use viridis_d with discrete data
txsample <- subset(txhousing, city %in%
  c("Houston", "Fort Worth", "San Antonio", "Dallas", "Austin"))
(d <- ggplot(data = txxsample, aes(x = sales, y = median)) +
  geom_point(aes(colour = city)))
d + scale_colour_viridis_d()

# Change scale label
d + scale_colour_viridis_d("City\nCenter")

# Select palette to use, see ?scales::viridis_pal for more details
d + scale_colour_viridis_d(option = "plasma")
d + scale_colour_viridis_d(option = "inferno")

# scale_fill_viridis_d works just the same as
# scale_colour_viridis_d but for fill colours
p <- ggplot(txamp, aes(x = median, fill = city)) +
  geom_histogram(position = "dodge", binwidth = 15000)
p + scale_fill_viridis_d()
# the order of colour can be reversed
p + scale_fill_viridis_d(direction = -1)

# Use viridis_c with continuous data
(v <- ggplot(faithful) +
  geom_tile(aes(waiting, eruptions, fill = density)))
v + scale_fill_viridis_c()
v + scale_fill_viridis_c(option = "plasma")

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())

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())

scale_x_log10(...)

scale_y_log10(...)

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 `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

**breaks**
One of:
- `NULL` for no breaks
- `waiver()` for the default breaks computed by the transformation object
- A numeric vector of positions
- A function that takes the limits as input and returns breaks as output

**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**
Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. Use the convenience function `expand_scale()` to generate the values for the expand argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

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

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

**trans**
Either the name of a transformation object, or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "exp", "identity", "log", "log10", "log1p", "log2", "logit", "probability", "probit", "reciprocal", "reverse" and "sqrt". A transformation object bundles together a transform, it’s inverse, and methods for generating breaks and labels. Transformation objects are defined in the scales package, and are called name_trans, e.g. `scales::boxcox_trans()`. You can create your own transformation with `scales::trans_new()`.

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

**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)
)
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:
pl + scale_y_log10()
pl + scale_y_sqrt()
pl + scale_y_reverse()

# Or you can supply a transformation in the `trans` argument:
pl + scale_y_continuous(trans = scales::reciprocal_trans())

# You can also create your own. See ?scales::trans_new

## scale_date

### Position scales for date/time data

**Description**

These are the default scales for the three date/time class. These will usually be added automatically. To override manually, use `scale_*_date` for dates (class `date`), `scale_*_datetime` for datetimes (class `POSIXct`), and `scale_*_time` for times (class `hms`).

**Usage**

```
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")

dscales_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")

dscales_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")

dscales_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")

dscales_x_time(name = waiver(), breaks = waiver(),
    minor_breaks = waiver(),
    labels = waiver(), limits = NULL, expand = waiver(), oob = censor,
    na.value = NA_real_, position = "bottom")

dscales_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 `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

**breaks**  
One of:  
- `NULL` for no breaks  
- `waiver()` for the breaks specified by `date_breaks`  
- A `Date/POSIXct` vector giving positions of breaks  
- 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 breaks  
- `waiver()` for the breaks specified by `date_minor_breaks`  
- A `Date/POSIXct` vector giving positions of minor breaks  
- A function that takes the limits as input and returns minor breaks as output

**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**  
Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. Use the convenience function `expand_scale()` to generate the values for the expand argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

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

**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. These scales will not produce a legend unless you also supply the breaks, labels, and type of guide you want.

Usage

```r
scale_colour_identity(..., guide = "none", aesthetics = "colour")
scale_fill_identity(..., guide = "none", aesthetics = "fill")
scale_shape_identity(..., guide = "none")
scale_linetype_identity(..., guide = "none")
scale_alpha_identity(..., guide = "none")
scale_size_identity(..., guide = "none")
scale_discrete_identity(aesthetics, ..., guide = "none")
scale_continuous_identity(aesthetics, ..., guide = "none")
```
scale_identity

Arguments

... Other arguments passed on to discrete_scale() or continuous_scale()
guide Guide to use for this scale. Defaults to "none".
aesthetics Character string or vector of character strings listing the name(s) of the aesthetic(s) that this scale works with. This can be useful, for example, to apply colour settings to the colour and fill aesthetics at the same time, via aesthetics = c("colour", "fill").

Details

The functions scale_colour_identity(), scale_fill_identity(), scale_size_identity(), etc. work on the aesthetics specified in the scale name: colour, fill, size, etc. However, the functions scale_colour_identity() and scale_fill_identity() also have an optional aesthetics argument that can be used to define both colour and fill aesthetic mappings via a single function call. The functions scale_discrete_identity() and scale_continuous_identity() are generic scales that can work with any aesthetic or set of aesthetics provided via the aesthetics argument.

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
```
scale_linetype

Scale for line patterns

Description

Default line types based on a set supplied by Richard Pearson, University of Manchester. Continuous values can not be mapped to line types.

Usage

scale_linetype(..., na.value = "blank")
scale_linetype_continuous(...)
scale_linetype_discrete(..., na.value = "blank")

Arguments

... Arguments passed on to discrete_scale
breaks One of:
  • NULL for no breaks
  • waiver() for the default breaks computed by the transformation object
  • A character vector of breaks
  • A function that takes the limits as input and returns breaks as output
limits A character vector that defines possible values of the scale and their order.
drop Should unused factor levels be omitted from the scale? The default, TRUE, uses the levels that appear in the data; FALSE uses all the levels in the factor.
na.translate Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify na.translate = FALSE.
aesthetics The names of the aesthetics that this scale works with
scale_name The name of the scale
palette A palette function that when called with a single integer argument (the number of levels in the scale) returns the values that they should take
name The name of the scale. Used as axis or legend title. If waiver(), the default, the name of the scale is taken from the first mapping used for that aesthetic. If NULL, the legend title will be omitted.
labels One of:
  • NULL for no labels
  • waiver() for the default labels computed by the transformation object
scale_manual

- A character vector giving labels (must be same length as breaks)
- A function that takes the breaks as input and returns labels as output

**guide** A function used to create a guide or its name. See `guides()` for more info.

**super** The super class to use for the constructed scale

**na.value** The linetype to use for NA values.

**Examples**

```r
def_lines <- data.frame(
  linetype = factor(1:4,
    labels = c("solid", "longdash", "dashed", "dotted")
  )
)
ggplot(df_lines) +
  geom_line(aes(linetype = linetype, yintercept = 0), size = 2) +
  scale_linetype_identity() +
  facet_grid(linetype ~ .) +
  theme_void(20)
```

**Description**

These functions allow you to specify your own set of mappings from levels in the data to aesthetic values.

**Usage**

```r
scale_colour_manual(..., values, aesthetics = "colour")

scale_fill_manual(..., values, aesthetics = "fill")

scale_size_manual(..., values)

scale_shape_manual(..., values)

scale_linetype_manual(..., values)
```
scale_alpha_manual(..., values)

scale_discrete_manual(aesthetics, ..., values)

Arguments

... Arguments passed on to discrete_scale

breaks One of:
  • NULL for no breaks
  • waiver() for the default breaks computed by the transformation object
  • A character vector of breaks
  • A function that takes the limits as input and returns breaks as output

limits A character vector that defines possible values of the scale and their order.

drop Should unused factor levels be omitted from the scale? The default, TRUE, uses the levels that appear in the data; FALSE uses all the levels in the factor.

na.translate Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify na.translate = FALSE.

na.value If na.translate = TRUE, what value aesthetic value should missing be displayed as? Does not apply to position scales where NA is always placed at the far right.

scale_name The name of the scale

palette A palette function that when called with a single integer argument (the number of levels in the scale) returns the values that they should take

name The name of the scale. Used as axis or legend title. If waiver(), the default, the name of the scale is taken from the first mapping used for that aesthetic. If NULL, the legend title will be omitted.

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

guide A function used to create a guide or its name. See guides() for more info.

super The super class to use for the constructed scale

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.

aesthetics Character string or vector of character strings listing the name(s) of the aesthetic(s) that this scale works with. This can be useful, for example, to apply colour settings to the colour and fill aesthetics at the same time, via aesthetics = c("colour", "fill").
Details

The functions `scale_colour_manual()`, `scale_fill_manual()`, `scale_size_manual()`, etc. work on the aesthetics specified in the scale name: colour, fill, size, etc. However, the functions `scale_colour_manual()` and `scale_fill_manual()` also have an optional aesthetics argument that can be used to define both colour and fill aesthetic mappings via a single function call (see examples). The function `scale_discrete_manual()` is a generic scale that can work with any aesthetic or set of aesthetics provided via the aesthetics argument.

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)

# You can set color and fill aesthetics at the same time
ggplot(
  mtcars,
  aes(mpg, wt, colour = factor(cyl), fill = factor(cyl))
) +
  geom_point(shape = 21, alpha = 0.5, size = 2) +
  scale_colour_manual(
    values = cols,
    aesthetics = c("colour", "fill")
  )

# 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"))
```

Description

`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**

```
scale_shape(..., solid = TRUE)
```

**Arguments**

```
...  Arguments passed on to discrete_scale

breaks One of:
   • NULL for no breaks
   • waiver() for the default breaks computed by the transformation object
   • A character vector of breaks
   • A function that takes the limits as input and returns breaks as output

limits A character vector that defines possible values of the scale and their order.

drop Should unused factor levels be omitted from the scale? The default, TRUE, uses the levels that appear in the data; FALSE uses all the levels in the factor.

na.translate Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify `na.translate = FALSE`.

na.value If `na.translate = TRUE`, what value aesthetic value should missing be displayed as? Does not apply to position scales where NA is always placed at the far right.

aesthetics The names of the aesthetics that this scale works with

scale_name The name of the scale

palette A palette function that when called with a single integer argument (the number of levels in the scale) returns the values that they should take

name The name of the scale. Used as axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If NULL, the legend title will be omitted.

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

guide A function used to create a guide or its name. See `guides()` for more info.

super The super class to use for the constructed scale

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

dsmall <- diamonds[sample(nrow(diamonds), 100), ]
(d <- ggplot(dsmall, aes(carat, price)) + geom_point(aes(shape = cut)))
d + scale_shape(solid = TRUE) # the default
d + scale_shape(solid = FALSE)
d + scale_shape(name = "Cut of diamond")

# To change order of levels, change order of
# underlying factor
levels(dsmall$cut) <- c("Fair", "Good", "Very Good", "Premium", "Ideal")

# 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 Scales for area or radius

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 waiver(), the default, the name of the scale is taken from the first mapping used for that aesthetic. If NULL, the legend title will be omitted.

breaks One of:
- NULL for no breaks
- waiver() for the default breaks computed by the transformation object
- A numeric vector of positions
- A function that takes the limits as input and returns breaks as output

**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, it's inverse, and methods for generating breaks and labels. Transformation objects are defined in the scales package, and are called name_trans, e.g. scales::boxcox_trans(). You can create your own transformation with scales::trans_new().

**guide**

A function used to create a guide or its name. See guides() for more info.

... Arguments passed on to continuous_scale

**name**

The name of the scale. Used as axis or legend title. If waiver(), the default, the name of the scale is taken from the first mapping used for that aesthetic. If NULL, the legend title will be omitted.

**breaks**

One of:
- NULL for no breaks
- waiver() for the default breaks computed by the transformation object
- A numeric vector of positions
- A function that takes the limits as input and returns breaks as output

**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.
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, it's inverse, and methods for generating breaks and labels. Transformation objects are defined in the scales package, and are called name_trans, e.g. scales::boxcox_trans(). You can create your own transformation with scales::trans_new().

guide A function used to create a guide or its name. See guides() for more info.

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

super The super class to use for the constructed scale

expand Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. Use the convenience function expand_scale() to generate the values for the expand argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

max_size Size of largest points.

See Also

scale_size_area() if you want 0 values to be mapped to points with size 0.

Examples

```r
p <- ggplot(mpg, aes(displ, hwy, size = hwy)) +
  geom_point()
p
p + scale_size("Highway mpg")
p + scale_size(range = c(0, 10))

# If you want zero value to have zero size, use scale_size_area:
p + scale_size_area()

# This is most useful when size is a count
ggplot(mpg, aes(class, cyl)) +
  geom_count() +
  scale_size_area()

# If you want to map size to radius (usually bad idea), use scale_radius
p + scale_radius()
```
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

Arguments passed on to discrete_scale

breaks One of:
  • NULL for no breaks
  • waiver() for the default breaks computed by the transformation object
  • A character vector of breaks
  • A function that takes the limits as input and returns breaks as output

limits A character vector that defines possible values of the scale and their order.

drop Should unused factor levels be omitted from the scale? The default, TRUE, uses the levels that appear in the data; FALSE uses all the levels in the factor.

na.translate Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify na.translate = FALSE.

na.value If na.translate = TRUE, what value aesthetic value should missing be displayed as? Does not apply to position scales where NA is always placed at the far right.

aesthetics The names of the aesthetics that this scale works with

scale_name The name of the scale

palette A palette function that when called with a single integer argument (the number of levels in the scale) returns the values that they should take

name The name of the scale. Used as axis or legend title. If waiver(), the default, the name of the scale is taken from the first mapping used for that aesthetic. If NULL, the legend title will be omitted.

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

**guide** A function used to create a guide or its name. See `guides()` for more info.

**super** The super class to use for the constructed scale

**expand** Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. Use the convenience function `expand_scale()` to generate the values for the expand argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

**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

```r
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("F1", "IF")

# See `reorder` to reorder based on the values of another variable
ggplot(mpg, aes(manufacturer, cty)) + geom_point()

ggplot(mpg, aes(reorder(manufacturer, cty), cty)) + geom_point()

ggplot(mpg, aes(reorder(manufacturer, displ), cty)) + geom_point()

# Use abbreviate as a formatter to reduce long names
ggplot(mpg, aes(reorder(manufacturer, displ), cty)) +
geom_point() +
scale_x_discrete(labels = abbreviate)
```
seals

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()
sec_axis

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 provide as a shorthand for creating a secondary axis that is a duplication of the primary
axis, effectively mirroring the primary axis.

Examples

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)
### Description

Most aesthetics are mapped from variables found in the data. Sometimes, however, you want to map from variables computed by the aesthetic. The most common example of this is the height of bars in `geom_histogram()`: the height does not come from a variable in the underlying data, but is instead mapped to the count computed by `stat_bin()`. The `stat()` function is a flag to ggplot2 to it that you want to use calculated aesthetics produced by the statistic.

### Usage

```
stat(x)
```

### Arguments

- **x**: An aesthetic expression using variables calculated by the stat.

### Details

This replaces the older approach of surrounding the variable name with ...  

### Examples

```r
# Default histogram display
ggplot(mpg, aes(displ)) +
  geom_histogram(aes(y = stat(count)))

# Scale tallest bin to 1
ggplot(mpg, aes(displ)) +
  geom_histogram(aes(y = stat(count / max(count))))
```

---

### 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

    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. It can also be a named logical vector to finely select the aesthetics to display.

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

Computed variables

  x x in data
  y cumulative density corresponding 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`: 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.
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.

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.

The number of segments to be used in drawing the ellipse.

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

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

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

References


Examples

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()

ggplot(faithful, aes(waiting, eruptions, fill = eruptions > 3)) +
stat_ellipse(geom = "polygon")
stat_function

Compute function for each x value

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.

- **...**: 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**: 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. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. borders().

Aesthetics

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

• group
• y

Learn more about setting these aesthetics in vignette("ggplot2-specs")

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

ggplot(data.frame(x = c(0, 2)), aes(x)) +
  stat_function(fun = exp, geom = "line")

# Specify range of x-axis
# Examples adapted from Kohske Takahashi

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
# Examples adapted from Kohske Takahashi

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
# Examples adapted from Kohske Takahashi
test <- function(x) (x ^ 2 + x + 20)
f + stat_function(fun = test)
stat_identity

Leave data as is

Description

The identity statistic leaves the data unchanged.

Usage

\[
\text{stat_identity}(\text{mapping} = \text{NULL}, \text{data} = \text{NULL}, \text{geom} = "\text{point}"), \text{position} = "\text{identity}"), \ldots, \text{show.legend} = \text{NA}, \text{inherit.aes} = \text{TRUE})
\]

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

Examples

\[
p <- \text{ggplot(mtcars, aes(wt, mpg)).Tru}\]
\[
p + \text{stat_identity()}
\]
**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**

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. It can also be a named logical vector to finely select the aesthetics to display.

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

Aesthetics

- 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 operates 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.ymax = NULL, fun.ymin = NULL, fun.args = list(), bins = 30,
binwidth = NULL, breaks = NULL, na.rm = FALSE, show.legend = NA,
inherit.aes = TRUE)

stat_summary(mapping = NULL, data = NULL, geom = "pointrange",
position = "identity", ..., fun.data = NULL, fun.y = NULL,
fun.ymax = NULL, fun.ymin = 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.
Alternatively, supply three individual functions that are each passed a vector of x’s and should return a single number.

Optional additional arguments passed on to the functions.

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

The width of the bins. Can be specified as a numeric value, or a function that calculates width from x. The default is to use 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.

Alternatively, you can supply a numeric vector giving the bin boundaries. Overrides binwidth, bins, center, and boundary.

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

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

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

Aesthetics

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

- x
- y
- group

Learn more about setting these aesthetics in vignette("ggplot2-specs")

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

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:
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")

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**

```r
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. It can also be a named logical vector to finely select the aesthetics to display.

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

Aesthetics

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

• group

Learn more about setting these aesthetics in vignette("ggplot2-specs")

Examples

ggplot(mtcars, aes(vs, am)) + geom_point(alpha = 0.1)

```r
ggplot(mtcars, aes(vs, am)) + geom_point(alpha = 0.1, stat = "unique")
```
Examples

```r
p <- ggplot(mpg, aes(displ, hwy)) + geom_point() + facet_wrap(~class)
b <- ggplot_build(p)

summarise_layout(b)
summarise_coord(b)
summarise_layers(b)
```

**theme**

*Modify components of a theme*

**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 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.y, axis.title.y.left, axis.title.y.right, axis.text, axis.text.x,
      axis.text.x.top, axis.text.x.bottom, axis.text.y, axis.text.y.left, axis.text.y.right,
      axis.ticks, axis.ticks.x, axis.ticks.x.top, axis.ticks.x.bottom, axis.ticks.y,
      axis.ticks.y.left, axis.ticks.y.right, axis.ticks.length, axis.line,
      axis.line.x, axis.line.x.top, axis.line.x.bottom, axis.line.y,
      axis.line.y.left, axis.line.y.right, 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.tag, plot.tag.position, plot.margin, strip.background,
      strip.background.x, strip.background.y, 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)
axis.title.x.bottom  x axis label on bottom axis (element_text; inherits from axis.title.x)
axis.title.y  y axis label (element_text; inherits from axis.title)
axis.title.y.left  y axis label on left axis (element_text; inherits from axis.title.y)
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.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.x.bottom  x axis tick labels on bottom axis (element_text; inherits from axis.text.x)
axis.text.y  y axis tick labels (element_text; inherits from axis.text)
axis.text.y.left  y axis tick labels on left axis (element_text; inherits from axis.text.y)
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.x.top  x axis tick marks on top axis (element_line; inherits from axis.ticks.x)
axis.ticks.x.bottom  x axis tick marks on bottom axis (element_line; inherits from axis.ticks.x)
axis.ticks.y  y axis tick marks (element_line; inherits from axis.ticks)
axis.ticks.y.left  y axis tick marks on left axis (element_line; inherits from axis.ticks.y)
axis.ticks.y.right  y axis tick marks on right axis (element_line; inherits from axis.ticks.y)
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.x.top  line along x axis on top axis (element_line; inherits from axis.line.x)
theme

axis.line.x.bottom
    line along x axis on bottom axis (element_line; inherits from axis.line.x)
axis.line.y
    line along y axis (element_line; inherits from axis.line)
axis.line.y.left
    line along y axis on left axis (element_line; inherits from axis.line.y)
axis.line.y.right
    line along y axis on right axis (element_line; inherits from axis.line.y)
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()
theme

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)

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.tag
upper-left label to identify a plot (text appearance) (element_text; inherits from title) left-aligned by default

plot.tag.position
The position of the tag as a string ("topleft", "top", "topright", "left", "right", "bottomleft", "bottom", "bottomright") or a coordinate. If a string, extra space will be added to accommodate the tag.
Theme

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 background x

Background of horizontal facet labels (element_rect; inherits from strip.background)

Strip background y

Background of vertical facet labels (element_rect; inherits from strip.background)

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_grey(). 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
pl + theme(plot.title = element_text(size = rel(2)))
pl + 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(wt, mpg)) +
    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

```r
# legend.justification sets the corner that the position refers to
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()
p

# 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"))
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.

```r
TXhousing
```

**Description**

Information about the housing market in Texas provided by the TAMU real estate center, [http://recenter.tamu.edu/](http://recenter.tamu.edu/).

**Usage**

```r
TXhousing
```

**Format**

A data frame with 8602 observations and 9 variables:

- **city**: Name of MLS area
- **year,month,date**: 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.

**Description**

Just like `aes()`, `vars()` is a quoting function that takes inputs to be evaluated in the context of a dataset. These inputs can be:

- variable names
- complex expressions

In both cases, the results (the vectors that the variable represents or the results of the expressions) are used to form faceting groups.

**Usage**

```r
vars(...)```
Arguments

Variables or expressions automatically quoted. These are evaluated in the context of the data to form faceting groups. Can be named (the names are passed to a labeller).

See Also

aes(), facet_wrap(), facet_grid()

Examples

```r
p <- ggplot(mtcars, aes(wt, disp)) + geom_point()
p + facet_wrap(vars(vs, am))

# vars() makes it easy to pass variables from wrapper functions:
wrap_by <- function(...) {
  facet_wrap(vars(...), labeller = label_both)
}
p + wrap_by(vs)
p + wrap_by(vs, am)

# You can also supply expressions to vars(). In this case it's often a
# good idea to supply a name as well:
p + wrap_by(drat = cut_number(drat, 3))

# Let's create another function for cutting and wrapping a
# variable. This time it will take a named argument instead of dots,
# so we'll have to use the "enquote and unquote" pattern:
wrap_cut <- function(var, n = 3) {
  # Let's enquote the named argument `var` to make it auto-quoted:
  var <- enquo(var)

  # `quo_name()` will create a nice default name:
  nm <- quo_name(var)

  # Now let's unquote everything at the right place. Note that we also
  # unquote `n` just in case the data frame has a column named
  # `n`. The latter would have precedence over our local variable
  # because the data is always masking the environment.
  wrap_by(list(nm := cut_number(nm, n)))
}

# Thanks to tidy eval idioms we now have another useful wrapper:
p + wrap_cut(drat)
```
Index

*Topic datasets
   diamonds, 33
   economics, 34
   faithfuld, 40
   ggsf, 116
   luv_colours, 136
   midwest, 139
   mpg, 140
   msleep, 141
   presidential, 149
   seals, 184
   txhousing, 208

*Topic hplot
   print.ggplot, 150
   +.gg, 5
   +.gg(), 204, 207
   %+%(+.gg), 5
   %+replace%(theme_get), 206
   %+replace%, 204

aes, 7
aes(), 6, 8, 9, 22, 41, 43, 46, 48, 49, 53, 55, 57, 60, 62, 64, 67, 69, 73, 74, 76, 80, 83, 85, 88, 91, 93, 95, 97, 99, 101, 104, 107, 109, 116, 187, 188, 190, 192, 193, 195, 198, 208, 209
aes_, 8
aes_(), 22, 41, 43, 46, 48, 49, 53, 55, 57, 60, 62, 64, 67, 69, 73, 74, 76, 80, 83, 85, 88, 91, 93, 95, 97, 99, 101, 104, 107, 109, 116, 187, 188, 190, 192, 193, 195, 198
aes_colour_fill_alpha, 10
aes_group_order, 11
aes_linetype_size_shape, 12
aes_position, 14
aes_q(aes_), 8
aes_string(aes_), 8
annotate, 15
annotate(), 77
annotation_custom, 16
annotation_logticks, 17
annotation_map, 19
annotation_raster, 19
as_labeller(), 129, 132
autolayer, 20
autolayer(), 21
autoplot, 21
autoplot(), 21
base::strwrap(), 132
borders, 21
borders(), 22, 44, 47, 48, 50, 53, 55, 58, 60, 63, 65, 68, 70, 73, 75, 77, 81, 83, 86, 89, 91, 93, 95, 98, 100, 102, 105, 108, 110, 117, 187, 189, 191, 192, 194, 196, 199
boxplot.stats(), 50
bquote(), 132
color(aes_colour_fill_alpha), 10
colors(), 136
colour(aes_colour_fill_alpha), 10
continuous_scale(), 154, 155, 165, 173
coord_cartesian, 23
coord_cartesian(), 27, 29, 30, 135
coord_equal(coord_fixed), 24
coord_fixed, 24
coord_flip, 25
coord_map, 26
coord_polar, 28
coord_quickmap(coord_map), 26
coord_sf(ggsf), 116
coord_trans, 30
coord_trans(), 18
CoordSF(ggsf), 116
cut_interval, 32
cut_number(cut_interval), 32
cut_width(cut_interval), 32
density(), 60, 110
derive(sec_axis), 184
diamonds, 33
discrete_scale(), 154, 155, 165, 173
dup_axis(sec_axis), 184
economics, 34
economics_long(economics), 34
element_blank(margin), 136
element_blank(), 204
element_line(margin), 136
element_line(), 204
element_rect(margin), 136
element_rect(), 204
element_text(margin), 136
element_text(), 122, 123, 125, 204
expand_limits, 34
expand_limits(), 48, 135
expand_scale, 35
expand_scale(), 159, 162, 164, 168, 171, 181, 183

facet_grid, 36
facet_grid(), 38, 129, 152, 209
facet_wrap, 38
facet_wrap(), 132, 152, 209
faithful, 40
faithfuldf, 40
fill(aes_colour_fill_alpha), 10
format.ggproto(print.ggproto), 151
fortify, 40
fortify(), 19, 21, 22, 41, 44, 46, 48, 49, 53, 55, 57, 60, 62, 64, 68, 69, 73, 74, 77, 80, 81, 83, 86, 88, 91, 93, 95, 97, 99, 101, 104, 107, 109, 112, 117, 187, 188, 190, 192, 193, 195, 198
fortify.lm(), 41

gem_col, 43
gem_col(geom_bar), 43
gem_contour, 52
gem_contour(), 63
gem_count, 54
gem_count(), 85, 86
gem_crossbar, 57
gem_crossbar(), 197
gem_curve(geom_segment), 101
gem_density, 59
gem_density(), 62, 109
gem_density2d(geom_density_2d), 62
gem_density2d(), 86
gem_density_2d, 62
gem_density_2d(), 54, 86
gem_dotplot, 64
gem_errorbar(geom_crossbar), 57
gem_errorbar(), 67, 197
gem_errorbarh, 67
gem_errorbarh(), 58
gem_freqpoly, 69
gem_freqpoly(), 61
gem_hex, 72
gem_hex(), 86
gem_histogram(geom_freqpoly), 69
gem_histogram(), 44, 45, 61, 186
gem_hline(geom_abline), 41
gem_jitter, 74
gem_jitter(), 51, 85
gem_label, 76
gem_label(), 184, 190, 192, 193, 195, 198

gem_point, 85
gem_point(), 46, 54, 76
gem_pointrange(geom_crossbar), 57
gem_pointrange(), 197
gem_polygon, 88
gem_polygon(), 84, 98
gem_qq(geom_qq_line), 90
gem_qq_line, 90
gem_quantile, 92
gem_quantile(), 51, 86
gem_raster, 94
median_hilow (hmisc), 128
mgcv::gam(), 104
midwest, 139
mpg, 140
msleep, 141
options(), 157
plot(), 151
plot.gaplot (print.ggplot), 150
png(), 115
position_dodge, 142, 144, 146, 147
position_dodge(), 44, 45
position_dodge2 (position_dodge), 142
position_dodge2(), 44, 45
position_fill (position_stack), 147
position_fill(), 44
position_identity, 142, 143, 144, 146, 147
position_jitter, 142, 144, 146, 147
position_jitterdodge, 142, 144, 145, 146, 147
position_nudge, 142, 144, 146, 147
position_stack, 142, 144, 146, 147
position_stack(), 44, 98
predict(), 105
presidential, 149
print.ggplot, 150
print.ggproto, 151
qplot, 151
quantreg::rq(), 93
quasiquotation, 7
quickplot (qplot), 151
quoting function, 7, 208
RColorBrewer::brewer_pal(), 155
rel (margin), 136
rescale(), 153, 160, 166
resolution, 153
scale_*_continuous, 35
scale_*_discrete, 35
scale_alpha, 154, 156, 160, 162, 164, 166
scale_alpha_continuous (scale_alpha), 154
scale_alpha_date (scale_alpha), 154
scale_alpha_datetime (scale_alpha), 154
scale_alpha_discrete (scale_alpha), 154
scale_alpha_identity (scale_identity), 172
scale_alpha_manual (scale_manual), 175
scale_alpha_ordinal (scale_alpha), 154
scale_color_brewer
(scale_colour_brewer), 155
scale_color_continuous
(scale_colour_gradient), 158
scale_color_discrete
(scale_colour_hue), 163
scale_color_distiller
(scale_colour_brewer), 155
scale_color_gradient
(scale_colour_gradient), 158
scale_color_gradient2
(scale_colour_gradient), 158
scale_color_gradientn
(scale_colour_gradient), 158
scale_color_gray (scale_colour_gray), 161
scale_color_hue (scale_colour_hue), 163
scale_color_identity (scale_identity), 172
scale_color_manual (scale_manual), 175
scale_color_viridis_c
(scale_colour_viridis_d), 165
scale_color_viridis_d
(scale_colour_viridis_d), 165
scale_colours_brewer, 154, 155, 160, 162, 164, 166
scale_colour_continuous, 157
scale_colour_date
(scale_colour_gradient), 158
scale_colour_datetime
(scale_colour_gradient), 158
scale_colour_discrete
(scale_colour_hue), 163
scale_colour_distiller
(scale_colour_brewer), 155
scale_colour_gradient, 154, 156, 158, 162, 164, 166
scale_colour_gradient2
(scale_colour_gradient), 158
scale_colour_gradient2()
(scale_colour_gradient), 159
scale_colour_gradientn
(scale_colour_gradient), 158
scale_colour_gradientn()
(scale_colour_gradient), 159
scale_colour_gray, 154, 156, 160, 161, 164, 166
scale_colour_hue, 154, 156, 160, 162, 163, 166
scale_colour_identity (scale_identity), 172
scale_colour_manual (scale_manual), 175
scale_colour_ordinal
  (scale_colour_viridis_d), 165
scale_colour_viridis_c
  (scale_colour_viridis_d), 165
scale_colour_viridis_c()
  (scale_colour_viridis_d), 157
scale_colour_viridis_d, 154, 156, 160, 162, 164, 165
scale_continuous, 167
scale_continuous_identity
  (scale_identity), 172
scale_date, 170
scale_discrete_identity
  (scale_identity), 172
scale_discrete_manual (scale_manual), 175
scale_fill_brewer
  (scale_colour_brewer), 155
scale_fill_continuous
  (scale_colour_continuous), 157
scale_fill_date
  (scale_colour_gradient), 158
scale_fill_datetime
  (scale_colour_gradient), 158
scale_fill_discrete (scale_colour_viridis_d), 157
scale_fill_distiller
  (scale_colour_brewer), 155
scale_fill_gradient
  (scale_colour_gradient), 158
scale_fill_gradient()
  (scale_colour_gradient), 157
scale_fill_gradient2
  (scale_colour_gradient), 158
scale_fill_gradientn
  (scale_colour_gradient), 158
scale_fill_grey (scale_colour_viridis_d), 161
scale_fill_hue (scale_colour_viridis_d), 163
scale_fill_identity (scale_identity), 172
scale_fill_manual (scale_manual), 175
scale_fill_ordinal
  (scale_colour_viridis_d), 165
scale_fill_viridis_c
  (scale_colour_viridis_d), 165
scale_fill_viridis_c()
  (scale_colour_viridis_d), 157
scale_fill_viridis_d
  (scale_colour_viridis_d), 165
scale_identity, 172
scale_linetype, 174
scale_linetype_continuous
  (scale_linetype), 174
scale_linetype_discrete
  (scale_linetype), 174
scale_linetype_identity
  (scale_identity), 172
scale_linetype_manual (scale_manual), 175
scale_manual, 175
scale_radius, 179
scale_shape, 177
scale_shape_continuous (scale_shape), 177
scale_shape_discrete (scale_shape), 177
scale_shape_identity (scale_identity), 172
scale_shape_manual (scale_manual), 175
scale_shape_manual()
  (scale_manual), 178
scale_shape_ordinal (scale_shape), 177
scale_size, 179
scale_size_area (scale_size), 179
scale_size_area()
  (scale_size), 181
scale_size_continuous (scale_size), 179
scale_size_date (scale_size), 179
scale_size_datetime (scale_size), 179
scale_size_discrete (scale_size), 179
scale_size_identity (scale_identity), 172
scale_size_manual (scale_manual), 175
scale_size_ordinal (scale_size), 179
scale_x_continuous, 172, 183
scale_x_continuous (scale_continuous), 167
scale_x_date, 169, 183
scale_x_date (scale_date), 170
scale_x_datetime (scale_date), 170
scale_x_discrete, 169, 172, 182
scale_x_log10 (scale_continuous), 167
scale_x_reverse (scale_continuous), 167
scale_x_sqrt (scale_continuous), 167
scale_x_time (scale_date), 170
scale_y_continuous (scale_continuous), 167
scale_y_continuous(), 18
scale_y_date(scale_date), 170
scale_y_datetime(scale_date), 170
scale_y_discrete(scale_x_discrete), 182
scale_y_log10(scale_continuous), 167
scale_y_log10(), 18
scale_y_reverse(scale_continuous), 167
scale_y_sqrt(scale_continuous), 167
scale_y_time(scale_date), 170
scales::boxcox_trans(), 159, 168, 180, 181
scales::seq_gradient_pal(), 160
scales::trans_new(), 30, 159, 168, 180, 181
seals, 184
sec_axis, 184
sec_axis(), 169
shape(aes_linetype_size_shape), 12
size(aes_linetype_size_shape), 12
stat, 186
stat_bin, 71
stat_bin(geom_freqpoly), 69
stat_bin(), 45, 186, 195
stat_bin2d(geom_bin2d), 46
stat_bin2d(), 74, 194
stat_bin_hex(geom_bin2d), 46
stat_binhex(geom_hex), 72
stat_binhex(geom_hex), 72
stat_binhex(), 47
stat_boxplot(geom_boxplot), 49
stat_contour(geom_contour), 52
stat_contour(), 63
stat_count(geom_bar), 43
stat_count(), 70, 71
stat_density(geom_density), 59
stat_density(), 111
stat_density2d(geom_density_2d), 62
stat_density_2d(geom_density_2d), 62
stat_ecdf, 186
stat_ellipse, 188
stat_function, 190
stat_identity, 192
stat_qq(geom_qq_line), 90
stat_qq_line(geom_qq_line), 90
stat_quantile(geom_quantile), 92
stat_sf(ggsf), 116
stat_smooth(geom_smooth), 103
stat_spoke(geom_spoke), 107
stat_sum(geom_count), 54
stat_summary(stat_summary_bin), 195
stat_summary(), 58, 128, 138, 193
stat_summary2d(stat_summary_2d), 193
stat_summary_2d, 193
stat_summary_2d(), 193
stat_summary_bin, 195
stat_summary_hex(stat_summary_2d), 193
stat_summary_hex(), 194
stat_unique, 198
stat_ydensity(geom_violin), 109
stats::bw.nrd(), 60, 110
StatF(ggsf), 116
strftime(), 171
substitute(), 9
summarise_coord(summarise_plot), 199
summarise_layers(summarise_plot), 199
summarise_layout(summarise_plot), 199
summarise_plot, 199
theme, 136, 200
theme(), 6, 119, 122, 123, 125, 126
theme_bw(ggtheme), 119
theme_classic(ggtheme), 119
theme_dark(ggtheme), 119
theme_get, 206
theme_gray(ggtheme), 119
theme_grey(ggtheme), 119
theme_light(ggtheme), 119
theme_linedraw(ggtheme), 119
theme_minimal(ggtheme), 119
theme_replace(theme_get), 206
theme_set(theme_get), 206
theme_test(ggtheme), 119
theme_update(theme_get), 206
theme_update(), 200
theme_update(), 206
theme_void(ggtheme), 119
txhousing, 208
vars, 208
vars(), 7, 36, 38
waiver(), 122, 125
x(aes_position), 14
xend(aes_position), 14
xlab(labs), 134
xlim(lims), 135
xmax(aes_position), 14
xmin (aes_position), 14
y (aes_position), 14
yend (aes_position), 14
ylab (labs), 134
ylim (lims), 135
ymax (aes_position), 14
ymin (aes_position), 14