# Package 'animint2'

July 18, 2019

**Title** Animated Interactive Grammar of Graphics

**Version** 2019.7.3

**URL** `https://github.com/tdhock/animint2`

**BugReports** `https://github.com/tdhock/animint2/issues`

**Description** Functions are provided for defining animated, interactive data visualizations in R code, and rendering on a web page. The 2018 Journal of Computational and Graphical Statistics paper, <doi:10.1080/10618600.2018.1513367> describes the concepts implemented.

**Depends** R (>= 3.4.0)

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

**Suggests** knitr (>= 1.5.33), servr, gistr (>= 0.2), shiny, htmltools, rmarkdown, testthat, devtools, httr, maps, ggplot2movies, hexbin, Hmisc, lattice, mapproj, maptools, mgcv, nlme, quantreg, rpart, svglite

**Enhances** sp

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**Collate** 'gganimintproto.r' 'aaa-.r' 'aes-calculated.r' 'aes-colour-fill-alpha.r' 'aes-group-order.r' 'aes-linetype-size-shape.r' 'aes-position.r' 'utilities.r' 'aes.r' 'legend-draw.r' 'geom-.r' 'annotation-custom.r' 'annotation-logticks.r' 'geom-polygon.r' 'geom-map.r' 'annotation-map.r' 'geom-raster.r' 'annotation-raster.r' 'annotation.r' 'autoplot.r' 'bench.r' 'bin.R' 'coord-.r' 'coord-cartesian-.r' 'coord-fixed.r' 'coord-flip.r' 'coord-map.r' 'coord-munch.r' 'coord-polar.r' 'coord-quickmap.R' 'coord-transform.r' 'data.R' 'facet-.r' 'facet-grid-.r' 'facet-labels.r' 'facet-layout.r'

'facet-locate.r' 'facet-null.r' 'facet-viewports.r'
'facet-wrap.r' 'fortify-lm.r' 'fortify-map.r'
'fortify-spatial.r' 'fortify.r' 'stat-.r' 'geom-abline.r'
'geom-rect.r' 'geom-bar.r' 'geom-bin2d.r' 'geom-blank.r'
'geom-boxplot.r' 'geom-path.r' 'geom-contour.r' 'geom-count.r'
'geom-crossbar.r' 'geom-segment.r' 'geom-curve.r'
'geom-defaults.r' 'geom-ribbon.r' 'geom-density.r'
'geom-density2d.r' 'geom-dotplot.r' 'geom-errorbar.r'
'geom-errorbarh.r' 'geom-freqpoly.r' 'geom-hex.r'
'geom-histogram.r' 'geom-hline.r' 'geom-jitter.r'
'geom-label.R' 'geom-linerange.r' 'geom-point.r'
'geom-pointrange.r' 'geom-quantile.r' 'geom-rug.r'
'geom-smooth.r' 'geom-spoke.r' 'geom-text.r' 'geom-tile.r'
'geom-violin.r' 'geom-vline.r' 'ggplot2.r' 'grob-absolute.r'
'grob-dotstack.r' 'grob-null.r' 'grouping.r' 'guide-colorbar.r'
'guide-legend.r' 'guides-.r' 'guides-axis.r' 'guides-grid.r'
'hexbin.R' 'labels.r' 'layer.r' 'limits.r' 'margins.R'
'panel.r' 'plot-build.r' 'plot-construction.r' 'plot-last.r'
'plot.r' 'position-.r' 'position-collide.r' 'position-dodge.r'
'position-fill.r' 'position-identity.r' 'position-jitter.r'
'position-jitterdodge.R' 'position-nudge.R' 'position-stack.r'
'quick-plot.r' 'range.r' 'save.r' 'scale-.r' 'scale-alpha.r'
'scale-brewer.r' 'scale-continuous.r' 'scale-date.r'
'scale-discrete-.r' 'scale-gradient.r' 'scale-grey.r'
'scale-hue.r' 'scale-identity.r' 'scale-linetype.r'
'scale-manual.r' 'scale-shape.r' 'scale-size.r' 'scale-type.R'
'scales-.r' 'stat-bin.r' 'stat-bin2d.r' 'stat-bindot.r'
'stat-binhex.r' 'stat-boxplot.r' 'stat-contour.r'
'stat-count.r' 'stat-density-2d.r' 'stat-density.r'
'stat-ecdf.r' 'stat-ellipse.R' 'stat-function.r'
'stat-identity.r' 'stat-qq.r' 'stat-quantile.r'
'stat-smooth-methods.r' 'stat-smooth.r' 'stat-sum.r'
'stat-summary-2d.r' 'stat-summary-bin.R' 'stat-summary-hex.r'
'stat-summary.r' 'stat-unique.r' 'stat-ydensity.r' 'summary.r'
'theme-defaults.r' 'theme-elements.r' 'theme.r'
'translate-qplot-ggplot.r' 'translate-qplot-lattice.r'
'utilities-break.r' 'utilities-grid.r' 'utilities-help.r'
'utilities-matrix.r' 'utilities-resolution.r'
'utilities-table.r' 'uu_zxx.r' 'uu_zzz.r' 'z_animint.R'
'z_animintHelpers.R' 'z_facets.R' 'z_geoms.R' 'z_gist.R'
'z_helperFunctions.R' 'z_knitr.R' 'z_print.R' 'z_scales.R'
'z_theme_animint.R' 'z_transformShape.R'

**RoxygenNote** 6.1.1

**NeedsCompilation** no

**Author** Toby Hocking [aut, cre] (Original animint code),
Hadley Wickham [aut] (Forked ggplot2 code),
Winston Chang [aut] (Forked ggplot2 code),
RStudio [cph] (Forked ggplot2 code),

Nicholas Lewin-Koh [aut] (hexGrob),
Martin Maechler [aut] (hexGrob),
Randall Prium [aut] (cut_width),
Susan VanderPlas [aut] (Animint GSOC 2013),
Carson Sievert [aut] (Animint GSOC 2014),
Kevin Ferris [aut] (Animint GSOC 2015),
Tony Tsai [aut] (Animint GSOC 2015),
Faizan Khan [aut] (Animint GSOC 2016-2017),
Vivek Kumar [aut] (Animint GSOC 2018)

**Maintainer** Toby Hocking <toby.hocking@r-project.org>

**Repository** CRAN

**Date/Publication** 2019-07-18 06:36:12 UTC

# R topics documented:

---

`+.gganimint` *Add a new component to a ggplot or theme object.*

---

## Description

This operator allows you to add objects to a ggplot or theme object.

## Usage

```
## S3 method for class 'gganimint'
e1 + e2

e1 %+% e2

e1 %+replace% e2
```

## Arguments

| | |
|---|---|
| `e1` | An object of class `ggplot` or `theme` |
| `e2` | A component to add to `e1` |

## Details

If the first object is an object of class `ggplot`, you can add the following types of objects, and it will return a modified ggplot object.

- `data.frame`: replace current data.frame (must use `%+%`)
- `uneval`: replace current aesthetics
- `layer`: add new layer
- `theme`: update plot theme
- `scale`: replace current scale
- `coord`: override current coordinate system
- `facet`: override current coordinate faceting

If the first object is an object of class `theme`, you can add another theme object. This will return a modified theme object.

For theme objects, the + operator and the `%+replace%` can be used to modify elements in themes.

The + operator 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, the `%+replace%` operator 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.

## See Also

`theme`

## Examples

```
### Adding objects to a ggplot object
p <- ggplot(mtcars, aes(wt, mpg, colour = disp)) +
  geom_point()

p
p + coord_cartesian(ylim = c(0, 40))
p + scale_colour_continuous(breaks = c(100, 300))
p + guides(colour = "colourbar")

# Use a different data frame
m <- mtcars[1:10, ]
p %+% m

### Adding objects to a theme object
# Compare these results of adding theme objects to other theme objects
add_el <- theme_grey() + theme(text = element_text(family = "Times"))
rep_el <- theme_grey() %+replace% theme(text = element_text(family = "Times"))

add_el$text
rep_el$text
```

---

# addShowSelectedForLegend

*Add a showSelected aesthetic if legend is specified*

---

## Description

Add a showSelected aesthetic if legend is specified

## Usage

```
addShowSelectedForLegend(meta, legend, L)
```

### Arguments

| | |
|---|---|
| meta | meta object with all information |
| legend | legend to scan for showSelected |
| L | layer of the plot |

### Value

L : Layer with additional mapping to new aesthetic

---

## addSSandCSasAesthetics

*Add the showSelected/clickSelects params to the aesthetics mapping*

---

### Description

Add the showSelected/clickSelects params to the aesthetics mapping

### Usage

```
addSSandCSasAesthetics(aesthetics, extra_params)
```

### Arguments

| | |
|---|---|
| aesthetics | list. Original aesthetics mapping of the layer |
| extra_params | named list containing the details of showSelected and clickSelects values of the layer |

### Details

Used before calling ggplot_build in parsePlot and while checking animint extensions to raise error

### Value

Modified aesthetics list with showSelected/clickSelects params added

## add_theme *Modify properties of an element in a theme object*

### Description

Modify properties of an element in a theme object

### Usage

```
add_theme(t1, t2, t2name)
```

### Arguments

| | |
|---|---|
| t1 | A theme object |
| t2 | A theme object that is to be added to t1 |
| t2name | A name of the t2 object. This is used for printing informative error messages. |

## aes *Define aesthetic mappings.*

### Description

Generate aesthetic mappings that describe how variables in the data are mapped to visual properties (aesthetics) of geoms. This function also standardise aesthetic names by performs partial name matching, converting color to colour, and old style R names to ggplot names (eg. pch to shape, cex to size)

### 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 can be omitted (because they are so common); all other aesthetics must be named. |

### See Also

See aes_q/aes_string for standard evaluation versions of aes.

See aes_colour_fill_alpha, aes_group_order, aes_linetype_size_shape and aes_position for more specific examples with different aesthetics.

## Examples

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

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

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

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

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

---

# aes_ *Define aesthetic mappings from strings, or quoted calls and formulas.*

---

## 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_`)

## 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

It's better to use `aes_q()`, because there's no easy way to create the equivalent to `aes(colour = "my colour")` or `aes{x = `X$1`}` with `aes_string()`.

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().

## 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))
```

---

# aes_colour_fill_alpha

*Colour related aesthetics: colour, fill and alpha*

---

## Description

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

## Examples

```
# 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
c + geom_bar(colour = "red")
# Combining both, you can see the changes more clearly
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
yrng <- range(economics$unemploy)
j <- j + geom_rect(aes(NULL, NULL, xmin = start, xmax = end, fill = party),
ymin = yrng[1], ymax = yrng[2], data = presidential)
j
j + scale_fill_manual(values = alpha(c("blue", "red"), .3))
```

---

`aes_group_order` *Aesthetics: group*

---

## Description

Aesthetics: group

## 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.

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

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

# Using linetypes
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, ..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, value01)) +
  geom_line(aes(linetype = variable))

# Size examples
# Should be specified with a numerical value (in millimetres),
# or from a variable source
p <- ggplot(mtcars, aes(wt, mpg))
```

```
p + geom_point(size = 4)
p + geom_point(aes(size = qsec))
p + geom_point(size = 2.5) +
  geom_hline(yintercept = 25, size = 3.5)

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

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

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

---

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

---

## Description

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

## Examples

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

```
# Using annotate
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)
```

---

`animint` *Create an animint*

---

## Description

Create an animated, interactive data visualization. The easiest way to get started is by reading the Animint2 Manual, http://members.cbio.mines-paristech.fr/~thocking/animint2-manual/Ch02-ggplot2.html

## Usage

```
animint(...)
```

## Arguments

| | |
|---|---|
| `...` | ggplots and options |

## Details

This function creates a list with the items in ... and attaches the animint class. It also provides default names for un-named ggplots. The list should contain ggplots and options. Each geom can be made interactive by using the showSelected and clickSelects parameters; each should be a character vector of selection variable names. For example geom_line(clickSelects="country") means that clicking the line changes the value of the "country" selection variable; geom_point(showSelected="year") means to only show the subset of data for the currently selected year.

## Value

list of class animint

## Author(s)

Toby Dylan Hocking

## Examples

```
library(animint2)
data(WorldBank, package="animint2")
years <- unique(WorldBank[, "year", drop=FALSE])
y1960 <- subset(WorldBank, year==1960)
animint(
  title="Linked scatterplot and time series", #web page title.
  time=list(variable="year",ms=3000), #variable and time delay used for animation.
  duration=list(year=1000), #smooth transition duration in milliseconds.
  selector.types=list(country="multiple"), #single/multiple selection for each variable.
  first=list( #selected values to show when viz is first rendered.
    country=c("Canada", "Japan"),
    year=1970),
  ## ggplots are rendered together for an interactive data viz.
  ts=ggplot()+
    theme_animint(width=500)+
    make_tallrect(WorldBank, "year")+
    geom_text(aes(
      year, life.expectancy, label=country),
      showSelected="country",
      clickSelects="country",
      hjust=1,
      data=y1960)+
    scale_x_continuous(limits=c(1950, NA))+
    geom_line(aes(
      year, life.expectancy, group=country, color=region),
      clickSelects="country",
      data=WorldBank,
      size=4,
      alpha=0.55),
  scatter=ggplot()+
    geom_point(aes(
      fertility.rate, life.expectancy,
      key=country, colour=region, size=population),
      showSelected="year",
      clickSelects="country",
      data=WorldBank)+
    geom_text(aes(
      fertility.rate, life.expectancy,
      key=country,
      label=country),
      showSelected=c("country", "year"),
      data=WorldBank)+
    geom_text(aes(
      5, 80, key=1, label=paste("year =", year)),
      showSelected="year",
      data=years)+
    scale_size_animint(pixel.range=c(2,20), breaks=10^(4:9)))
```

`animint2dir` *Compile and render an animint in a local directory.*

## Description

This function converts an animint plot.list into a directory of files which can be used to render the interactive data visualization in a web browser.

## Usage

```
animint2dir(plot.list, out.dir = NULL, json.file = "plot.json",
  open.browser = interactive(), css.file = "")
```

## Arguments

| | |
|---|---|
| `plot.list` | a named list of ggplots and option lists. |
| `out.dir` | directory to store html/js/csv files. If it exists already, it will be removed before writing the new directory/files. |
| `json.file` | character string that names the JSON file with metadata associated with the plot. |
| `open.browser` | Should R open a browser? If yes, be sure to configure your browser to allow access to local files, as some browsers block this by default (e.g. chrome). |
| `css.file` | character string for non-empty css file to include. Provided file will be copied to the output directory as styles.css |

## Value

invisible list of ggplots in list format.

## Examples

```
## Make a Gapminder plot (aka Google motion chart), which is actually
## just a scatterplot with size and color that moves over time.
library(animint2)
data(WorldBank)
gapminder <-
  list(title="Linked scatterplot and time series",
       ts=ggplot()+
       make_tallrect(WorldBank, "year")+
       geom_line(aes(year, life.expectancy, group=country, color=region),
                 clickSelects="country",
                 data=WorldBank, size=4, alpha=3/5),
       time=list(variable="year",ms=3000),
       duration=list(year=1000),
       scatter=ggplot()+
       geom_point(aes(fertility.rate, life.expectancy,
                      key=country, colour=region, size=population),
                  showSelected="year",
                  clickSelects="country",
```

```
                  data=WorldBank)+
        geom_text(aes(fertility.rate, life.expectancy, label=country),
                  showSelected=c("country", "year"),
                  data=WorldBank)+
        make_text(WorldBank, 5, 80, "year")+
        scale_size_animint(pixel.range=c(2,20), breaks=10^(4:9)))
animint2dir(gapminder)

data(worldPop)
## Linked bar and line plots of world population by subcontinent,
## inspired by polychartjs.
popPlots <-
  list(bars=ggplot()+
       geom_bar(aes(x=subcontinent, y=population),
                clickSelects="subcontinent",
                showSelected="year",
                data=worldPop, stat="identity", position="identity")+
       ## This make_text creates a geom_text that shows the current
       ## selected value of the year variable.
       make_text(worldPop, 1, 3e6, "year")+
       coord_flip(),
       lines=ggplot()+
       ## This make_tallrect tiles the background of the lineplot with
       ## rects that can be clicked to select the year variable.
       make_tallrect(worldPop, "year")+
       ## This geom_point does not have aes(clickSelects) so its alpha
       ## transparency behaves normally: all points have alpha=1/4.
       geom_point(aes(year, population, colour=type),
                  data=worldPop, size=4, alpha=1/4)+
       ## This geom_line DOES have aes(clickSelects) so only the
       ## selected line has the specified alpha=3/4. The other
       ## unselected lines have 0.5 less (alpha=1/4).
       geom_line(aes(year, population, group=subcontinent),
                 clickSelects="subcontinent",
                 data=worldPop, size=4, alpha=3/4))
animint2dir(popPlots)
## Make it animated by specifying year as the variable to animate and
## an interval of 2000 milliseconds between animation frames.
popAnim <- c(popPlots, list(time=list(variable="year",ms=2000)))
animint2dir(popAnim)
## Make the animation smooth by specifying a duration of 1000 ms for
## geoms with aes(showSelected=year).
popSmooth <- c(popAnim, list(duration=list(year=1000)))
animint2dir(popSmooth)
```

---

`animint2gist` *Convert a list of ggplots to an interactive animation and post files as a gist*

---

### Description

Before using this function set your appropriate 'github.username' and 'github.password' options

### Usage

```
animint2gist(plot.list, description = plot.list$title, browse = TRUE,
  ...)
```

### Arguments

| | |
|---|---|
| plot.list | a named list of ggplots and option lists. |
| description | Brief description of gist. This becomes the plot title on the bl.ocks/username page. |
| browse | logical. Prompt browser to view viz on http://bl.ocks.org |
| ... | options passed onto animint2dir and gistr::gist_create |

### Examples

```
## Not run:
library(animint)
iris$id <- 1:nrow(iris)
viz <- list(petal=ggplot()+
         geom_point(aes(Petal.Width, Petal.Length, fill=Species,
                        clickSelects=id), data=iris),
       sepal=ggplot()+
         geom_point(aes(Sepal.Width, Sepal.Length, fill=Species,
                        clickSelects=id), data=iris))
animint2gist(viz, description = "My animint plot")

## End(Not run)
```

---

## animintOutput *Shiny ui output function*

---

### Description

Shiny ui output function

### Usage

```
animintOutput(outputId)
```

### Arguments

| | |
|---|---|
| outputId | output variable to read the plot from |

### See Also

http://shiny.rstudio.com/articles/building-outputs.html

`annotate` *Create an annotation layer.*

## Description

This function adds geoms to a plot. 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), removes missing values with a warning. If `TRUE` silently removes missing values. |

## 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"))
```

annotation_custom *Annotation: Custom grob.*

## 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

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

## Arguments

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

## Details

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

## Note

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

## Examples

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

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

# Inset plot
df2 <- data.frame(x = 1 , y = 1)
g <- ggplotGrob(ggplot(df2, aes(x, y)) +
  geom_point() +
```

```
  theme(plot.background = element_rect(colour = "black")))
base +
  annotation_custom(grob = g, xmin = 1, xmax = 10, ymin = 8, ymax = 10)
```

---

annotation_logticks

*Annotation: log tick marks*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| base | the base of the log (default 10) |
| sides | a string that controls which sides of the plot the log ticks appear on. It can be set to a string containing any of "trbl", for top, right, bottom, and left. |
| 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 unit object specifying the length of the short tick marks |
| mid | a unit object specifying the length of the middle tick marks. In base 10, these are the "5" ticks. |
| long | a unit object specifying the length of the long tick marks. In base 10, these are the "1" (or "10") ticks. |
| colour | Colour of the tick marks. |
| size | Thickness of tick marks, in mm. |
| linetype | Linetype of tick marks (solid, dashed, etc.) |
| alpha | The transparency of the tick marks. |
| color | An alias for colour. |
| ... | Other parameters passed on to the layer |

## See Also

scale_y_continuous, scale_y_log10 for log scale transformations.

coord_trans for log coordinate transformations.

## Examples

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

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

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

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

b + annotation_logticks()

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

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

---

`annotation_map` *Annotation: maps.*

---

## Description

Annotation: maps.

### 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

```
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).

### 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. |

### Details

Most useful for adding bitmap images.

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

---

## as.list.gganimintproto

*Convert a gganimintproto object to a list*

---

### Description

This will not include the object's `super` member.

### Usage

```
## S3 method for class 'gganimintproto'
as.list(x, inherit = TRUE, ...)
```

### Arguments

| | |
|---|---|
| `x` | A gganimintproto object to convert to a list. |
| `inherit` | If `TRUE` (the default), flatten all inherited items into the returned list. If `FALSE`, do not include any inherited items. |
| `...` | Further arguments to pass to `as.list.environment`. |

`as_labeller` *Coerce to labeller function*

## Description

This transforms objects to labeller functions. Used internally by `labeller()`.

## Usage

```
as_labeller(x, default = label_value, multi_line = TRUE)
```

## Arguments

| | |
|---|---|
| `x` | Object to coerce to a labeller function. If a named character vector, it is used as a lookup table before being passed on to `default`. If a non-labeller function, it is assumed it takes and returns character vectors and is applied to the labels. If a labeller, it is simply applied to the labels. |
| `default` | Default labeller to process the labels produced by lookup tables or modified by non-labeller functions. |
| `multi_line` | Whether to display the labels of multiple factors on separate lines. This is passed to the labeller function. |

## See Also

`labeller()`, labellers

## Examples

```
p <- ggplot(mtcars, aes(disp, drat)) + geom_point()
p + facet_wrap(~am)

# Rename labels on the fly with a lookup character vector
to_string <- as_labeller(c(`0` = "Zero", `1` = "One"))
p + facet_wrap(~am, labeller = to_string)

# Quickly transform a function operating on character vectors to a
# labeller function:
appender <- function(string, suffix = "-foo") paste0(string, suffix)
p + facet_wrap(~am, labeller = as_labeller(appender))

# If you have more than one facetting variable, be sure to dispatch
# your labeller to the right variable with labeller()
p + facet_grid(cyl ~ am, labeller = labeller(am = to_string))
```

## autoplot *Create a complete ggplot appropriate to a particular data type*

### Description

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

### Usage

```
autoplot(object, ...)
```

### Arguments

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

### Value

a ggplot object

### See Also

`ggplot` and `fortify`

## borders *Create a layer of map borders.*

### Description

Create a layer of map borders.

### Usage

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

### Arguments

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

## 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)
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()
}
```

---

# breakpoints *The breakpointError of simulated signals*

---

## Description

Two noisy signals were sampled from a latent signal with known breakpoints, which were used to measure the error of estimated models with 1,...,20 segments.

## Usage

```
data(breakpoints)
```

## Format

A list of 5 data.frames: error contains the breakpointError of the estimated models, signals contains the noisy signals, breaks contains the breakpoints in the estimated signals, segments contains the estimated segments, and imprecision contains the normalized imprecision curves which were used to evaluate the error.

## Source

The breakpointError package was used to measure the model error, see etc/breakpoints.R.

## calc_element *Calculate the element properties, by inheriting properties from its parents*

### Description

Calculate the element properties, by inheriting properties from its parents

### Usage

```
calc_element(element, theme, verbose = FALSE)
```

### Arguments

| | |
|---|---|
| element | The name of the theme element to calculate |
| theme | A theme object (like theme_grey()) |
| verbose | If TRUE, print out which elements this one inherits from |

### Examples

```
t <- theme_grey()
calc_element('text', t)

# Compare the "raw" element definition to the element with calculated inheritance
t$axis.text.x
calc_element('axis.text.x', t, verbose = TRUE)

# This reports that axis.text.x inherits from axis.text,
# which inherits from text. You can view each of them with:
t$axis.text.x
t$axis.text
t$text
```

## checkAnimationTimeVar *Check animation variable for errors*

### Description

Check animation variable for errors

### Usage

```
checkAnimationTimeVar(timeVarList)
```

### Arguments

`timeVarList` `plot.list$time` in `animint2dir` to check for errors

### Value

`NULL` :Stops with an error for invalid input

---

## checkExtraParams *Check* `extra_params` *argument for duplicates, non-named list*

---

### Description

Check `extra_params` argument for duplicates, non-named list

### Usage

```
checkExtraParams(extra_params, aes_mapping)
```

### Arguments

| | |
|---|---|
| `extra_params` | named list containing the details of showSelected and clickSelects values of the layer |
| `aes_mapping` | aesthetics mapping of the layer |

### Value

Modified `extra_params` list

---

## checkForSSandCSasAesthetics *Check if showSelected and clickSelects have been used as aesthetics as in old syntax. If yes, raise error*

---

### Description

Check if showSelected and clickSelects have been used as aesthetics as in old syntax. If yes, raise error

### Usage

```
checkForSSandCSasAesthetics(aesthetics, plot_name)
```

### Arguments

| | |
|---|---|
| `aesthetics` | list. aesthetics mapping of the layer |
| `plot_name` | character vector of the plot the layer belongs to |

### Value

NULL Throws error if used as aesthetics

---

## checkPlotForAnimintExtensions

*Performs error checking on the plot for animint extensions*

---

### Description

Performs error checking on the plot for animint extensions

### Usage

```
checkPlotForAnimintExtensions(p, plot_name)
```

### Arguments

| | |
|---|---|
| p | plot from plot.list to check for errors |
| plot_name | plot name error check. Should be alphanumeric and should begin with an alphabet |

### Value

NULL :Stops with an error for invalid input

---

## checkPlotList

*Check plot.list for errors*

---

### Description

Check that plot.list is a list and every element is named

### Usage

```
checkPlotList(plot.list)
```

### Arguments

| | |
|---|---|
| plot.list | from animint2dir to check for errors |

### Value

Throws an error for invalid values

checkSingleShowSelectedValue *Issue warnings for non interactive plots where there is only one showSelected value*

## Description

Issue warnings for non interactive plots where there is only one showSelected value

## Usage

```
checkSingleShowSelectedValue(selectors)
```

## Arguments

| | |
|---|---|
| selectors | selectors to check for warnings |

## Value

NULL

colsNotToCopy *Filter out columns that do not need to be copied*

## Description

Filter out columns that do not need to be copied

## Usage

```
colsNotToCopy(g, s.aes)
```

## Arguments

| | |
|---|---|
| g | Geom with columns |
| s.aes | Selector aesthetics |

## Value

Character vector of columns not to be copied

`coord_cartesian` *Cartesian coordinates.*

## Description

The Cartesian coordinate system is the most familiar, and common, type of coordinate system. Setting limits on the coordinate system will zoom the plot (like you're looking at it with a magnifying glass), and will not change the underlying data like setting limits on a scale will.

## Usage

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

## Arguments

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

## Examples

```
# 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)

# Simiarly, 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 relationship between x and y scales.*

---

## Description

A fixed scale coordinate system forces a specified ratio between the physical representation of data units on the axes. The ratio represents the number of units on the y-axis equivalent to one unit on the x-axis. The default, `ratio = 1`, ensures that one unit on the x-axis is the same length as one unit on the y-axis. Ratios higher than one make units on the y axis longer than units on the x-axis, and vice versa. This is similar to `eqscplot`, but it works for all types of graphics.

## Usage

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

## Arguments

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

## Examples

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

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

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

coord_flip *Flipped cartesian coordinates.*

## Description

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

## Usage

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

## Arguments

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

## Examples

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

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

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

The representation of a portion of the earth, which is approximately spherical, onto a flat 2D plane requires a projection. This is what coord_map does. These projections account for the fact that the actual length (in km) of one degree of longitude varies between the equator and the pole. Near the equator, the ratio between the lengths of one degree of latitude and one degree of longitude is approximately 1. Near the pole, it is tends towards infinity because the length of one degree of longitude tends towards 0. For regions that span only a few degrees and are not too close to the poles, setting the aspect ratio of the plot to the appropriate lat/lon ratio approximates the usual mercator projection. This is what coord_quickmap does. With coord_map all elements of the graphic have to be projected which is not the case here. So coord_quickmap has the advantage of being much faster, in particular for complex plots such as those using with geom_tile, at the expense of correctness in the projection. This coordinate system provides the full range of map projections available in the mapproj package.

## Usage

```
coord_map(projection = "mercator", ..., orientation = NULL,
  xlim = NULL, ylim = NULL)

coord_quickmap(xlim = NULL, ylim = NULL, expand = TRUE)
```

## Arguments

| | |
|---|---|
| projection | projection to use, see mapproject for list |
| ... | other arguments passed on to mapproject |
| orientation | projection orientation, which defaults to c(90,0,mean(range(x))). This is not optimal for many projections, so you will have to supply your own. See mapproject for more information. |
| xlim | manually specific x limits (in degrees of longitude) |
| ylim | manually specific y limits (in degrees of latitude) |
| expand | If TRUE, the default, adds a small expansion factor to the limits to ensure that data and axes don't overlap. If FALSE, limits are taken exactly from the data or xlim/ylim. |

## 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
nzmap + coord_map("cylindrical")
nzmap + coord_map("azequalarea", orientation = c(-36.92,174.6,0))

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))
}
```

---

`coord_polar` *Polar coordinates.*

---

## Description

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

## Usage

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

## Arguments

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

## Examples

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

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



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

# The bullseye chart
pie + coord_polar()

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

# Windrose + doughnut plot
if (require("ggplot2movies")) {
```

```
movies$rrating <- cut_interval(movies$rating, length = 1)
movies$budgetq <- cut_number(movies$budget, 4)

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

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

---

`coord_trans` *Transformed cartesian coordinate system.*

---

## Description

`coord_trans` is different to scale transformations in that it occurs after statistical transformation and will affect the visual appearance of geoms - there is no guarantee that straight lines will continue to be straight.

## Usage

```
coord_trans(x = "identity", y = "identity", limx = NULL,
  limy = NULL, xtrans, ytrans)
```

## Arguments

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

## Details

All current transformations only work with continuous values - see `trans_new` for list of transformations, and instructions on how to create your own.

## Examples

```
# See ?geom_boxplot for other examples

# Three ways of doing transformation in ggplot:
#  * by transforming the data
ggplot(diamonds, aes(log10(carat), log10(price))) +
  geom_point()
#  * by transforming the scales
ggplot(diamonds, aes(carat, price)) +
```

```
  geom_point() +
  scale_x_log10() +
  scale_y_log10()
#  * by transforming the coordinate system:
ggplot(diamonds, aes(carat, price)) +
  geom_point() +
  coord_trans(x = "log10", y = "log10")

# The difference between transforming the scales and
# transforming the coordinate system is that scale
# transformation occurs BEFORE statistics, and coordinate
# transformation afterwards.  Coordinate transformation also
# changes the shape of geoms:

d <- subset(diamonds, carat > 0.5)

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

ggplot(d, aes(carat, price)) +
  geom_point() +
  geom_smooth(method = "lm") +
  coord_trans(x = "log10", y = "log10")

# Here I used a subset of diamonds so that the smoothed line didn't
# drop below zero, which obviously causes problems on the log-transformed
# scale

# With a combination of scale and coordinate transformation, it's
# possible to do back-transformations:
ggplot(diamonds, aes(carat, price)) +
  geom_point() +
  geom_smooth(method = "lm") +
  scale_x_log10() +
  scale_y_log10() +
  coord_trans(x = scales::exp_trans(10), y = scales::exp_trans(10))

# cf.
ggplot(diamonds, aes(carat, price)) +
  geom_point() +
  geom_smooth(method = "lm")

# Also works with discrete scales
df <- data.frame(a = abs(rnorm(26)),letters)
plot <- ggplot(df,aes(a,letters)) + geom_point()

plot + coord_trans(x = "log10")
plot + coord_trans(x = "sqrt")
```

cut_interval *Cut up numeric vector into useful groups.*

## Description

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

## Usage

```
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 |
| length | length of each interval |
| ... | other arguments passed on to cut |
| width | The bin width. |
| center, boundary | Specify either the position of edge or the center of a bin. Since all bins are aligned, specifying the position of a single bin (which doesn't need to be in the range of the data) affects the location of all bins. If not specified, uses the "tile layers algorithm", and sets the boundary to half of the binwidth.<br>To center on integers, width = 1 and center = 0. boundary = 0.5. |
| closed | One of "right" or "left" indicating whether right or left edges of bins are included in the bin. |

## Author(s)

Randall Prium contributed most of the implementation of cut_width.

## See Also

cut_number

## Examples

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

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

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

---

# diamonds *Prices of 50,000 round cut diamonds*

---

## Description

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

## Usage

```
diamonds
```

## Format

A data frame with 53940 rows and 10 variables:

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

## economics *US economic time series.*

### Description

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

### Usage

```
economics

economics_long
```

### Format

A data frame with 478 rows and 6 variables

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

## element_blank *Theme element: blank. This theme element draws nothing, and assigns no space*

### Description

Theme element: blank. This theme element draws nothing, and assigns no space

### Usage

```
element_blank()
```

## element_line *Theme element: line.*

### Description

Theme element: line.

### Usage

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

### Arguments

| | |
|---|---|
| colour | line colour |
| size | line size |
| linetype | line type |
| lineend | line end |
| color | an alias for colour |

## element_rect *Theme element: rectangle.*

### Description

Most often used for backgrounds and borders.

### Usage

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

### Arguments

| | |
|---|---|
| fill | fill colour |
| colour | border colour |
| size | border size |
| linetype | border linetype |
| color | an alias for colour |

## element_text *Theme element: text.*

### Description

Theme element: text.

### Usage

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

### Arguments

| | |
|---|---|
| family | font family |
| face | font face ("plain", "italic", "bold", "bold.italic") |
| colour | text colour |
| size | text size (in pts) |
| hjust | horizontal justification (in [0, 1]) |
| vjust | vertical justification (in [0, 1]) |
| angle | angle (in [0, 360]) |
| lineheight | line height |
| color | an alias for colour |
| 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. |

## expand_limits *Expand the plot limits with data.*

### Description

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

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

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

---

## facet_grid *Lay out panels in a grid.*

---

### Description

Lay out panels in a grid.

### Usage

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

### Arguments

| | |
|---|---|
| facets | a formula with the rows (of the tabular display) on the LHS and the columns (of the tabular display) on the RHS; the dot in the formula is used to indicate there should be no faceting on this dimension (either row or column). The formula can also be provided as a string instead of a classical formula object |
| margins | either a logical value or a character vector. Margins are additional facets which contain all the data for each of the possible values of the faceting variables. If FALSE, no additional facets are included (the default). If TRUE, margins are included for all faceting variables. If specified as a character vector, it is the names of variables for which margins are to be created. |
| scales | Are scales shared across all facets (the default, "fixed"), or do they vary across rows ("free_x"), columns ("free_y"), or both rows and columns ("free") |

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

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

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

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

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

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

## Examples

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

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

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

# If you combine a facetted dataset with a dataset that lacks those
# facetting variables, the data will be repeated across the missing
# combinations:
df <- data.frame(displ = mean(mpg$displ), cty = mean(mpg$cty))
p +
  facet_grid(. ~ cyl) +
  geom_point(data = df, colour = "red", size = 2)

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

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

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

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

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

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

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

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

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

# Margins can be specified by logically (all yes or all no) or by specific
# variables as (character) variable names
mg <- ggplot(mtcars, aes(x = mpg, y = wt)) + geom_point()
mg + facet_grid(vs + am ~ gear)
mg + facet_grid(vs + am ~ gear, margins = TRUE)
mg + facet_grid(vs + am ~ gear, margins = "am")
# when margins are made over "vs", since the facets for "am" vary
# within the values of "vs", the marginal facet for "vs" is also
# a margin over "am".
mg + facet_grid(vs + am ~ gear, margins = "vs")
mg + facet_grid(vs + am ~ gear, margins = "gear")
mg + facet_grid(vs + am ~ gear, margins = c("gear", "am"))
```

## facet_null *Facet specification: a single panel.*

### Description

Facet specification: a single panel.

### Usage

```
facet_null(shrink = TRUE)
```

### Arguments

| | |
|---|---|
| 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. |

### Examples

```
# facet_null is the default facetting specification if you
# don't override it with facet_grid or facet_wrap
ggplot(mtcars, aes(mpg, wt)) + geom_point()
```

## facet_wrap *Wrap a 1d ribbon of panels into 2d.*

### Description

Most displays are roughly rectangular, so if you have a categorical variable with many levels, it doesn't make sense to try and display them all in one row (or one column). To solve this dilemma, facet_wrap wraps a 1d sequence of panels into 2d, making best use of screen real estate.

### Usage

```
facet_wrap(facets, nrow = NULL, ncol = NULL, scales = "fixed",
  shrink = TRUE, labeller = "label_value", as.table = TRUE,
  switch = NULL, drop = TRUE, dir = "h")
```

### Arguments

| | |
|---|---|
| facets | Either 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 of the plot. If switch is "x", they will be displayed to the bottom. If "y", they will be displayed to the left, near the y axis. |
| 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. |

## Examples

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

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


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

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

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

```
  facet_wrap(~class2)

# By default, the same scales are used for all panels. You can allow
# scales to vary across the panels with the `scales` argument.
# Free scales make it easier to see patterns within each panel, but
# harder to compare across panels.
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  facet_wrap(~class, scales = "free")

# To repeat the same data in every panel, simply construct a data frame
# that does not contain the facetting variable.
ggplot(mpg, aes(displ, hwy)) +
  geom_point(data = transform(mpg, class = NULL), colour = "grey85") +
  geom_point() +
  facet_wrap(~class)

# Use `switch` to display the facet labels near an axis, acting as
# a subtitle for this 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, switch = "x") +
  theme(strip.background = element_blank())
```

---

`faithfuld` *2d density estimate of Old Faithful data*

---

## Description

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

## Usage

```
faithfuld
```

## Format

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

## FluView *FluView*

### Description

Data about Flu outbreaks.

### Usage

```
data("FluView")
```

### Format

The format is a named list of two data.frames.

## format.gganimintproto *Format a gganimintproto object*

### Description

Format a gganimintproto object

### Usage

```
## S3 method for class 'gganimintproto'
format(x, ..., flat = TRUE)
```

### Arguments

| | |
|---|---|
| x | A gganimintproto object to print. |
| ... | If the gganimintproto 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. |

## fortify *Fortify a model with data.*

### Description

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

### Usage

```
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`

## fortify.lm *Supplement the data fitted to a linear model with model fit statistics.*

### Description

If you have missing values in your model data, you may need to refit the model with `na.action = na.exclude`.

### Usage

```
## S3 method for class 'lm'
fortify(model, data = model$model, ...)
```

### Arguments

| | |
|---|---|
| `model` | linear model |
| `data` | data set, defaults to data used to fit model |
| `...` | not used by this method |

## Value

The original data with extra columns:

| | |
|---|---|
| `.hat` | Diagonal of the hat matrix |
| `.sigma` | Estimate of residual standard deviation when corresponding observation is dropped from model |
| `.cooksd` | Cooks distance, `cooks.distance` |
| `.fitted` | Fitted values of model |
| `.resid` | Residuals |
| `.stdresid` | Standardised residuals |

## Examples

```
mod <- lm(mpg ~ wt, data = mtcars)
head(fortify(mod))
head(fortify(mod, mtcars))

plot(mod, which = 1)

ggplot(mod, aes(.fitted, .resid)) +
  geom_point() +
  geom_hline(yintercept = 0) +
  geom_smooth(se = FALSE)

ggplot(mod, aes(.fitted, .stdresid)) +
  geom_point() +
  geom_hline(yintercept = 0) +
  geom_smooth(se = FALSE)

ggplot(fortify(mod, mtcars), aes(.fitted, .stdresid)) +
  geom_point(aes(colour = factor(cyl)))

ggplot(fortify(mod, mtcars), aes(mpg, .stdresid)) +
  geom_point(aes(colour = factor(cyl)))

plot(mod, which = 2)
ggplot(mod) +
  stat_qq(aes(sample = .stdresid)) +
  geom_abline()

plot(mod, which = 3)
ggplot(mod, aes(.fitted, sqrt(abs(.stdresid)))) +
  geom_point() +
  geom_smooth(se = FALSE)

plot(mod, which = 4)
ggplot(mod, aes(seq_along(.cooksd), .cooksd)) +
  geom_bar(stat = "identity")

plot(mod, which = 5)
```

```
ggplot(mod, aes(.hat, .stdresid)) +
  geom_vline(size = 2, colour = "white", xintercept = 0) +
  geom_hline(size = 2, colour = "white", yintercept = 0) +
  geom_point() + geom_smooth(se = FALSE)

ggplot(mod, aes(.hat, .stdresid)) +
  geom_point(aes(size = .cooksd)) +
  geom_smooth(se = FALSE, size = 0.5)

plot(mod, which = 6)
ggplot(mod, aes(.hat, .cooksd)) +
  geom_vline(xintercept = 0, colour = NA) +
  geom_abline(slope = seq(0, 3, by = 0.5), colour = "white") +
  geom_smooth(se = FALSE) +
  geom_point()

ggplot(mod, aes(.hat, .cooksd)) +
  geom_point(aes(size = .cooksd / .hat)) +
  scale_size_area()
```

---

`fortify.map` *Fortify method for map objects.*

---

## Description

This function turns a map into a data frame that can more easily be plotted with ggplot2.

## Usage

```
## S3 method for class 'map'
fortify(model, data, ...)
```

## Arguments

| | |
|---|---|
| `model` | map object |
| `data` | not used by this method |
| `...` | not used by this method |

## See Also

`map_data` and `borders`

## Examples

```
if (require("maps")) {
ca <- map("county", "ca", plot = FALSE, fill = TRUE)
head(fortify(ca))
ggplot(ca, aes(long, lat)) +
  geom_polygon(aes(group = group))
```

```
tx <- map("county", "texas", plot = FALSE, fill = TRUE)
head(fortify(tx))
ggplot(tx, aes(long, lat)) +
  geom_polygon(aes(group = group), colour = "white")
}
```

---

`fortify.sp` *Fortify method for classes from the sp package.*

---

## Description

To figure out the correct variable name for region, inspect `as.data.frame(model)`.

## Usage

```
## S3 method for class 'SpatialPolygonsDataFrame'
fortify(model, data, region = NULL,
  ...)

## S3 method for class 'SpatialPolygons'
fortify(model, data, ...)

## S3 method for class 'Polygons'
fortify(model, data, ...)

## S3 method for class 'Polygon'
fortify(model, data, ...)

## S3 method for class 'SpatialLinesDataFrame'
fortify(model, data, ...)

## S3 method for class 'Lines'
fortify(model, data, ...)

## S3 method for class 'Line'
fortify(model, data, ...)
```

## Arguments

| | |
|---|---|
| `model` | `SpatialPolygonsDataFrame` to convert into a dataframe. |
| `data` | not used by this method |
| `region` | name of variable used to split up regions |
| `...` | not used by this method |

### Examples

```
if (require("maptools")) {
 sids <- system.file("shapes/sids.shp", package="maptools")
 nc1 <- readShapePoly(sids,
   proj4string = CRS("+proj=longlat +datum=NAD27"))
 nc1_df <- fortify(nc1)
}
```

---

## generation.loci *Evolution simulation*

---

### Description

Allele frequencies for 100 loci and 12 populations were simulated over 100 generations.

### Usage

```
data(generation.loci)
```

### Format

A data frame with 120000 observations on the following 4 variables.

`locus` a numeric vector

`population` a numeric vector

`generation` a numeric vector

`frequency` a numeric vector

`color` factor: blue, red, or neutral

`type` factor: balancing, none, or positive

### Source

Data generated using nicholsonppp::sim.drift.selection, see code in etc/generation.loci.R.

`geom_abline` *Lines: horizontal, vertical, and specified by slope and intercept.*

## Description

These paired geoms and stats add straight lines to a plot, either horizontal, vertical or specified by slope and intercept. These are useful for annotating plots.

## Usage

```
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:<br>If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot`.<br>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.<br>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), 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. |
| `xintercept, yintercept, slope, intercept` | Parameters that control the position of the line. If these are set, `data`, `mapping` and `show.legend` are overridden |

## Details

These geoms act slightly different to other geoms. You can supply the parameters in two ways: either as arguments to the layer function, or via aesthetics. If you use arguments, e.g. `geom_abline(intercept = 0, slope = 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

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

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

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

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

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

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

# You can also control other aesthetics
```

```
ggplot(mtcars, aes(mpg, wt, colour = wt)) +
  geom_point() +
  geom_hline(aes(yintercept = wt, colour = wt), mean_wt) +
  facet_wrap(~ cyl)
```

---

`geom_bar` *Bars, rectangles with bases on x-axis*

---

## Description

There are two types of bar charts, determined by what is mapped to bar height. By default, `geom_bar` uses `stat="count"` which makes the height of the bar proportion to the number of cases in each group (or if the `weight` aethetic is supplied, the sum of the weights). If you want the heights of the bars to represent values in the data, use `stat="identity"` and map a variable to the `y` aesthetic.

## Usage

```
geom_bar(mapping = NULL, data = NULL, stat = "count",
  position = "stack", ..., width = NULL, binwidth = 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:<br>If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot`.<br>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.<br>A `function` will be called with a single argument, the plot data. The return value must be a `data.frame.`, and will be used as the layer data. |
| `position` | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| `...` | other arguments passed on to `layer`. These are often aesthetics, used to set an aesthetic to a fixed value, like `color = "red"` or `size = 3`. They may also be parameters to the paired geom/stat. |
| `width` | Bar width. By default, set to 90% of the resolution of the data. |

| | |
|---|---|
| binwidth | geom_bar no longer has a binwidth argument - if you use it you'll get an warning telling to you use geom_histogram instead. |
| na.rm | If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |
| geom, stat | Override the default connection between geom_bar and stat_count. |

## Details

A bar chart maps the height of the bar to a variable, 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[1]. This is why it doesn't make sense to use a log-scaled y axis with a bar chart.

By default, multiple x's occurring in the same place will be stacked atop one another by position_stack. If you want them to be dodged side-to-side, see position_dodge. Finally, position_fill shows relative proportions at each x by stacking the bars and then stretching or squashing to the same height.

## Aesthetics

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

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

## Computed variables

**count** number of points in bin

**prop** groupwise proportion

## See Also

geom_histogram for continuous data, position_dodge for creating side-by-side barcharts.

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

[1] http://www.b-eye-network.com/view/index.php?cid=2468

## 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))

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

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


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

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

# To change plot order of bars, change levels in underlying factor
reorder_size <- function(x) {
  factor(x, levels = names(sort(table(x))))
}
ggplot(mpg, aes(reorder_size(class))) + geom_bar()
```

---

`geom_bin2d` *Add heatmap of 2d bin counts.*

---

## Description

Add heatmap of 2d bin counts.

## Usage

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

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:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>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), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |
| geom, 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_bin2dunderstands the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- fill

## 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 *Blank, draws nothing.*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `mapping` | Set of aesthetic mappings created by `aes` or `aes_`. If specified and `inherit.aes` = `TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping. |
| `data` | The data to be displayed in this layer. There are three options:<br>If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot`.<br>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.<br>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. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |

## Examples

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

---

# geom_boxplot *Box and whiskers plot.*

---

## Description

The lower and upper "hinges" correspond to the first and third quartiles (the 25th and 75th percentiles). This differs slightly from the method used by the boxplot function, and may be apparent with small samples. See boxplot.stats for for more information on how hinge positions are calculated for boxplot.

## Usage

```
geom_boxplot(mapping = NULL, data = NULL, stat = "boxplot",
  position = "dodge", ..., outlier.colour = NULL,
  outlier.color = NULL, outlier.shape = 19, outlier.size = 1.5,
  outlier.stroke = 0.5, 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:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>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.shape, outlier.size, outlier.stroke | Default aesthetics for outliers. Set to NULL to inherit from the aesthetics used for the box.<br>In the unlikely event you specify both US and UK spellings of colour, the US spelling will take precedence. |
| notch | if FALSE (default) make a standard box plot. If TRUE, make a notched box plot. Notches are used to compare groups; if the notches of two boxes do not overlap, this suggests that the medians are significantly different. |
| notchwidth | for a notched box plot, width of the notch relative to the body (default 0.5) |
| varwidth | if FALSE (default) make a standard box plot. If TRUE, boxes are drawn with widths proportional to the square-roots of the number of observations in the groups (possibly weighted, using the weight aesthetic). |
| na.rm | If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |
| 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 |

## Details

The upper whisker extends from the hinge to the highest value that is within 1.5 * IQR of 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 lowest value within 1.5 * IQR of the hinge. Data beyond the end of the whiskers are outliers and plotted as points (as specified by Tukey).

In a notched box plot, the notches extend 1.58 * IQR / sqrt(n). This gives a roughly 95 See McGill et al. (1978) for more details.

### Aesthetics

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

- **lower**
- **middle**
- **upper**
- **x**
- **ymax**
- **ymin**
- alpha
- colour
- fill
- linetype
- shape
- size
- weight

### Computed variables

**width** width of boxplot

**ymin** lower whisker = smallest observation greater than or equal to lower hinge - 1.5 * IQR

**lower** lower hinge, 25% quantile

**notchlower** lower edge of notch = median - 1.58 * IQR / sqrt(n)

**middle** median, 50% quantile

**notchupper** upper edge of notch = median + 1.58 * IQR / sqrt(n)

**upper** upper hinge, 75% quantile

**ymax** upper whisker = largest observation less than or equal to upper hinge + 1.5 * IQR

### References

McGill, R., Tukey, J. W. and Larsen, W. A. (1978) Variations of box plots. The American Statistician 32, 12-16.

### See Also

`stat_quantile` to view quantiles conditioned on a continuous variable, `geom_jitter` for another way to look at conditional distributions.

## Examples

```
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)))


# 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 *Display contours of a 3d surface in 2d.*

---

## Description

Display contours of a 3d surface in 2d.

## Usage

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

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

## Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by aes or aes_. If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |
| data | The data to be displayed in this layer. There are three options:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>A function will be called with a single argument, the plot data. The return value must be a data.frame., and will be used as the layer data. |
| stat | The statistical transformation to use on the data for this layer, as a string. |
| position | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| ... | other arguments passed on to layer. These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat. |
| lineend | Line end style (round, butt, square) |
| linejoin | Line join style (round, mitre, bevel) |
| linemitre | Line mitre limit (number greater than 1) |
| na.rm | If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |
| geom | The geometric object to use display the data |

## Aesthetics

geom_contourunderstands the following aesthetics (required aesthetics are in bold):

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

### Computed variables

**level** height of contour

### See Also

`geom_density_2d`: 2d density contours

### Examples

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

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


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

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

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

geom_count *Count the number of observations at each location.*

## Description

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

## Usage

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

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:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>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), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |
| 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
- shape
- size
- stroke

### Computed variables

**n** number of observations at position

**prop** percent of points in that panel at that position

### Examples

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

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

# Best used in conjunction with scale_size_area which ensures that
# counts of zero would be given size 0. Doesn't make much different
# here because the smallest count is already close to 0.
ggplot(mpg, aes(cty, hwy)) +
 geom_count()
 scale_size_area()

# Display proportions instead of counts -------------------------------------
# By default, all categorical variables in the plot form the groups.
# Specifying geom_count without a group identifier leads to a plot which is
# not useful:
d <- ggplot(diamonds, aes(x = cut, y = clarity))
d + geom_count(aes(size = ..prop..))
# To correct this problem and achieve a more desirable plot, we need
# to specify which group the proportion is to be calculated over.
d + geom_count(aes(size = ..prop.., group = 1)) +
  scale_size_area(max_size = 10)

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

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

## Description

Various ways of representing a vertical interval defined by `x`, `ymin` and `ymax`.

## Usage

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

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

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

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:<br>If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot`.<br>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.<br>A `function` will be called with a single argument, the plot data. The return value must be a `data.frame.`, and will be used as the layer data. |
| `stat` | The statistical transformation to use on the data for this layer, as a string. |
| `position` | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| `...` | other arguments passed on to `layer`. These are often aesthetics, used to set an aesthetic to a fixed value, like `color = "red"` or `size = 3`. They may also be parameters to the paired geom/stat. |

| | |
|---|---|
| fatten | A multiplicative factor used to increase the size of the middle bar in geom_crossbar() and the middle point in geom_pointrange(). |
| na.rm | If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |

## Aesthetics

geom_linerangeunderstands the following aesthetics (required aesthetics are in bold):

- **x**
- **ymax**
- **ymin**
- alpha
- colour
- linetype
- size

## See Also

stat_summary for examples of these guys in use, geom_smooth for continuous analog

## Examples

```
#' # Create a simple example dataset
df <- data.frame(
  trt = factor(c(1, 1, 2, 2)),
  resp = c(1, 5, 3, 4),
  group = factor(c(1, 2, 1, 2)),
  upper = c(1.1, 5.3, 3.3, 4.2),
  lower = c(0.8, 4.6, 2.4, 3.6)
)

p <- ggplot(df, aes(trt, resp, colour = group))
p + geom_linerange(aes(ymin = lower, ymax = upper))
p + geom_pointrange(aes(ymin = lower, ymax = upper))
p + geom_crossbar(aes(ymin = lower, ymax = upper), width = 0.2)
p + geom_errorbar(aes(ymin = lower, ymax = upper), width = 0.2)

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

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

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

---

`geom_density` *Display a smooth density estimate.*

---

## Description

A kernel density estimate, useful for display the distribution of variables with underlying smoothness.

## Usage

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

stat_density(mapping = NULL, data = NULL, geom = "area",
  position = "stack", ..., bw = "nrd0", adjust = 1,
  kernel = "gaussian", 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:<br>If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot`.<br>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.<br>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), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |
| geom, stat | Use to override the default connection between geom_density and stat_density. |
| bw | the smoothing bandwidth to be used, see density for details |
| adjust | adjustment of the bandwidth, see density for details |
| kernel | kernel used for density estimation, 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_densityunderstands the following aesthetics (required aesthetics are in bold):

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

## Computed variables

**density** density estimate

**count** density * number of points - useful for stacked density plots

**scaled** density estimate, scaled to maximum of 1

## See Also

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

### Examples

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

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

---

## geom_density_2d *Contours from a 2d density estimate.*

---

### Description

Perform a 2D kernel density estimation using kde2d and display the results with contours. This can be useful for dealing with overplotting.

### Usage

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

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:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>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), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |
| geom, stat | Use to override the default connection between geom_density_2d and stat_density_2d. |
| contour | If TRUE, contour the results of the 2d density estimation |
| n | number of grid points in each direction |
| h | Bandwidth (vector of length two). If NULL, estimated using bandwidth.nrd. |

## Aesthetics

geom_density_2dunderstands the following aesthetics (required aesthetics are in bold):

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

### Computed variables

Same as `stat_contour`

### See Also

`geom_contour` for contour drawing geom, `stat_sum` for another way of dealing with overplotting

### Examples

```
m <- ggplot(faithful, aes(x = eruptions, y = waiting)) +
 geom_point() +
 xlim(0.5, 6) +
 ylim(40, 110)
m + geom_density_2d()

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

set.seed(4393)
dsmall <- diamonds[sample(nrow(diamonds), 1000), ]
d <- ggplot(dsmall, aes(x, y))
# If you map an aesthetic to a categorical variable, you will get a
# set of contours for each value of that variable
d + geom_density_2d(aes(colour = cut))

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

---

## `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

```
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:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>A function will be called with a single argument, the plot data. The return value must be a data.frame., and will be used as the layer data. |
| position | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| ... | other arguments passed on to layer. These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat. |
| binwidth | When method is "dotdensity", this specifies maximum bin width. When method is "histodot", this specifies bin width. Defaults to 1/30 of the range of the data |
| binaxis | The axis to bin along, "x" (default) or "y" |
| method | "dotdensity" (default) for dot-density binning, or "histodot" for fixed bin widths (like stat_bin) |
| binpositions | When method is "dotdensity", "bygroup" (default) determines positions of the bins for each group separately. "all" determines positions of the bins with all the data taken together; this is used for aligning dot stacks across multiple groups. |
| stackdir | which direction to stack the dots. "up" (default), "down", "center", "centerwhole" (centered, but with dots aligned) |
| stackratio | how close to stack the dots. Default is 1, where dots just just touch. Use smaller values for closer, overlapping dots. |
| 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), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |

## Details

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

## 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

Wilkinson, L. (1999) Dot plots. The American Statistician, 53(3), 276-281.

## 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

Horizontal error bars

## Usage

```
geom_errorbarh(mapping = NULL, data = NULL, stat = "identity",
```

```
  position = "identity", ..., na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE)
```

## Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by aes or aes_. If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |
| data | The data to be displayed in this layer. There are three options:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>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), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |

## Aesthetics

geom_errorbarhunderstands the following aesthetics (required aesthetics are in bold):

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

## See Also

geom_errorbar: vertical error bars

## Examples

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

# Define the top and bottom of the errorbars

p <- ggplot(df, aes(resp, trt, colour = group))
p + geom_point() +
  geom_errorbarh(aes(xmax = resp + se, xmin = resp - se))
p + geom_point() +
  geom_errorbarh(aes(xmax = resp + se, xmin = resp - se, height = .2))
```

---

# geom_freqpoly *Histograms and frequency polygons.*

---

## Description

Display a 1d distribution by dividing into bins and counting the number of observations in each bin. Histograms use bars; frequency polygons use lines.

## Usage

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

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

stat_bin(mapping = NULL, data = NULL, geom = "bar",
  position = "stack", ..., binwidth = NULL, bins = NULL,
  center = NULL, boundary = NULL, 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:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>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), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |
| binwidth | The width of the bins. The default is to use bins bins that cover the range of the data. You should always override this value, exploring multiple widths to find the best to illustrate the stories in your data.<br>The bin width of a date variable is the number of days in each time; the bin width of a time variable is the number of seconds. |
| bins | Number of bins. Overridden by binwidth. Defaults to 30 |
| geom, stat | Use to override the default connection between geom_histogram/geom_freqpoly and stat_bin. |
| center | The center of one of the bins. Note that if center is above or below the range of the data, things will be shifted by an appropriate number of widths. To center on integers, for example, use width=1 and center=0, even if 0 is outside the range of the data. At most one of center and boundary may be specified. |
| boundary | A boundary between two bins. As with center, things are shifted when boundary is outside the range of the data. For example, to center on integers, use width = 1 and boundary = 0.5, even if 1 is outside the range of the data. At most one of center and boundary may be specified. |
| closed | One of "right" or "left" indicating whether right or left edges of bins are included in the bin. |
| pad | If TRUE, adds empty bins at either end of x. This ensures frequency polygons touch 0. Defaults to FALSE. |

## Details

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

### Aesthetics

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

### Computed variables

**count** number of points in bin

**density** density of points in bin, scaled to integrate to 1

**ncount** count, scaled to maximum of 1

**ndensity** density, scaled to maximum of 1

### See Also

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

### Examples

```
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)
ggplot(diamonds, aes(price, colour = cut)) +
  geom_freqpoly(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, ..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(origin = 0) + coord_trans(x = "log10")
# Use origin = 0, to make sure we don't take sqrt of negative values
m + geom_histogram(origin = 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()
}
rm(movies)
```

---

`geom_hex` *Hexagon binning.*

---

## Description

Hexagon binning.

## Usage

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

stat_bin_hex(mapping = NULL, data = NULL, geom = "hex",
  position = "identity", ..., bins = 30, binwidth = NULL,
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```

## Arguments

| | |
|---|---|
| `mapping` | Set of aesthetic mappings created by `aes` or `aes_`. If specified and `inherit.aes` = `TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping. |
| `data` | The data to be displayed in this layer. There are three options:<br>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.<br>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), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |
| 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_hexunderstands the following aesthetics (required aesthetics are in bold):

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

## 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))
```

---

`geom_jitter` *Points, jittered to reduce overplotting.*

---

## Description

The jitter geom is a convenient default for geom_point with position = 'jitter'. It's a useful way of handling overplotting caused by discreteness in smaller datasets.

## Usage

```
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:<br>If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot`.<br>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.<br>A `function` will be called with a single argument, the plot data. The return value must be a `data.frame.`, and will be used as the layer data. |
| `stat` | The statistical transformation to use on the data for this layer, as a string. |
| `position` | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| `...` | other arguments passed on to `layer`. These are often aesthetics, used to set an aesthetic to a fixed value, like `color = "red"` or `size = 3`. They may also be parameters to the paired geom/stat. |
| `width` | Amount of vertical and horizontal jitter. The jitter is added in both positive and negative directions, so the total spread is twice the value specified here.<br>If omitted, defaults to 40% of the resolution of the data: this means the jitter values will occupy 80% of the implied bins. Categorical data is aligned on the integers, so a width or height of 0.5 will spread the data so it's not possible to see the distinction between the categories. |

| | |
|---|---|
| height | Amount of vertical and horizontal jitter. The jitter is added in both positive and negative directions, so the total spread is twice the value specified here.<br>If omitted, defaults to 40% of the resolution of the data: this means the jitter values will occupy 80% of the implied bins. Categorical data is aligned on the integers, so a width or height of 0.5 will spread the data so it's not possible to see the distinction between the categories. |
| na.rm | If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |

## Aesthetics

geom_pointunderstands the following aesthetics (required aesthetics are in bold):

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

## See Also

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

## Examples

```
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()
ggplot(mpg, aes(cyl, hwy)) + geom_jitter(width = 0.25)

# 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)
```

geom_label *Textual annotations.*

## Description

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

## Usage

```
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:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>A function will be called with a single argument, the plot data. The return value must be a data.frame., and will be used as the layer data. |
| stat | The statistical transformation to use on the data for this layer, as a string. |
| position | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| ... | other arguments passed on to layer. These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat. |
| parse | If TRUE, the labels will be parsed into expressions and displayed as described in ?plotmath |
| nudge_x, nudge_y | Horizontal and vertical adjustment to nudge labels by. Useful for offsetting text from points, particularly on discrete scales. |

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

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

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

na.rm
: If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.

show.legend
: logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.

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

check_overlap
: If TRUE, text that overlaps previous text in the same layer will not be plotted. A quick and dirty way

## Details

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

## Aesthetics

geom_textunderstands the following aesthetics (required aesthetics are in bold):

- **label**
- **x**
- **y**
- alpha
- angle
- colour
- family
- fontface
- hjust
- lineheight
- size
- vjust

geom_label

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

### Alignment

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

### Examples

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

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

# Set aesthetics to fixed value
p + geom_point() + geom_text(hjust = 0, nudge_x = 0.05)
p + geom_point() + geom_text(vjust = 0, nudge_y = 0.5)
p + geom_point() + geom_text(angle = 45)
## Not run:
# Doesn't work on all systems
p + geom_text(family = "Times New Roman")

## End(Not run)

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

p + geom_text(aes(size = wt))
# Scale height of text, rather than sqrt(height)
p + geom_text(aes(size = wt)) + scale_radius(range = c(3,6))

# You can display expressions by setting parse = TRUE.  The
# details of the display are described in ?plotmath, but note that
# geom_text uses strings, not expressions.
p + geom_text(aes(label = paste(wt, "^(", cyl, ")", sep = "")),
  parse = TRUE)

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


# Aligning labels and bars ------------------------------------------------
df <- data.frame(
```

```
  x = factor(c(1, 1, 2, 2)),
  y = c(1, 3, 2, 1),
  grp = c("a", "b", "a", "b")
)

# ggplot2 doesn't know you want to give the labels the same virtual width
# as the bars:
ggplot(data = df, aes(x, y, fill = grp, label = y)) +
  geom_bar(stat = "identity", position = "dodge") +
  geom_text(position = "dodge")
# So tell it:
ggplot(data = df, aes(x, y, fill = grp, label = y)) +
  geom_bar(stat = "identity", position = "dodge") +
  geom_text(position = position_dodge(0.9))
# Use you can't nudge and dodge text, so instead adjust the y postion
ggplot(data = df, aes(x, y, fill = grp, label = y)) +
  geom_bar(stat = "identity", position = "dodge") +
  geom_text(aes(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 do the computation yourself
df <- transform(df, mid_y = ave(df$y, df$x, FUN = function(val) cumsum(val) - (0.5 * val)))

ggplot(data = df, aes(x, y, fill = grp, label = y)) +
 geom_bar(stat = "identity") +
 geom_text(aes(y = mid_y))

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

---

`geom_map` *Polygons from a reference map.*

---

## Description

Does not affect position scales.

## Usage

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

### Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by aes or aes_. If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |
| data | The data to be displayed in this layer. There are three options:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>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. |
| map | Data frame that contains the map coordinates. This will typically be created using fortify on a spatial object. It must contain columns x or long, y or lat, and region or id. |
| na.rm | If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |

### Aesthetics

geom_mapunderstands the following aesthetics (required aesthetics are in bold):

- **map_id**
- alpha
- colour
- fill
- linetype
- size

### Examples

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

```
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_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)
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.

## Usage

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

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

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

## Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by aes or aes_. If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |
| data | The data to be displayed in this layer. There are three options:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>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 arrow |
| na.rm | If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |
| direction | direction of stairs: 'vh' for vertical then horizontal, or 'hv' for horizontal then vertical |

### Aesthetics

geom_pathunderstands the following aesthetics (required aesthetics are in bold):

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

### See Also

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

### Examples

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

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

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

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

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

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

```
base + geom_path(size = 10, linejoin = "mitre", lineend = "butt")

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


# Setting line type vs colour/size
# Line type needs to be applied to a line as a whole, so it can
# not be used with colour or size that vary across a line
x <- seq(0.01, .99, length.out = 100)
df <- data.frame(
  x = rep(x, 2),
  y = c(qlogis(x), 2 * qlogis(x)),
  group = rep(c("a","b"),
  each = 100)
)
p <- ggplot(df, aes(x=x, y=y, group=group))
# These work
p + geom_line(linetype = 2)
p + geom_line(aes(colour = group), linetype = 2)
p + geom_line(aes(colour = x))
# But this doesn't
should_stop(p + geom_line(aes(colour = x), linetype=2))
```

---

`geom_point` *Points, as for a scatterplot*

---

## Description

The point geom is used to create scatterplots.

## Usage

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

## Arguments

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

| | |
|---|---|
| data | The data to be displayed in this layer. There are three options:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>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), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |

## Details

The scatterplot is useful for displaying the relationship between two continuous variables, although it can also be used with one continuous and one categorical variable, or two categorical variables. See geom_jitter for possibilities.

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 stat_sum. Another technique is to use transparent points, e.g. geom_point(alpha = 0.05).

## Aesthetics

geom_pointunderstands the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- alpha

- colour
- fill
- shape
- size
- stroke

## See Also

`scale_size` to see scale area of points, instead of radius, `geom_jitter` to jitter points to reduce (mild) overplotting

## Examples

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

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

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

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


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


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


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

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

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

---

`geom_polygon` *Polygon, a filled path.*

---

## Description

Polygon, a filled path.

## 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:<br>If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot`.<br>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.<br>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), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |

## Aesthetics

geom_polygonunderstands the following aesthetics (required aesthetics are in bold):

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

## 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, 1.2, 2.5, 1.1, 0.3,
  0.5, 1.2, 2.5, 1.2, 1.3, 2.7, 1.2, 0.5, 0.6, 1.3),
  y = c(-0.5, 0, 1, 0.5, 0, 0.5, 1.5, 1, 0.5, 1, 2.1, 1.7, 1, 1.5,
  2.2, 2.1, 1.7, 2.1, 3.2, 2.8, 2.1, 2.2, 3.3, 3.2)
)

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

```
(p <- ggplot(datapoly, aes(x = x, y = y)) + geom_polygon(aes(fill = value, group = id)))

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

stream <- data.frame(
  x = cumsum(runif(50, max = 0.1)),
  y = cumsum(runif(50,max = 0.1))
)

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

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

---

`geom_quantile` *Add quantile lines from a quantile regression.*

---

## Description

This can be used as a continuous analogue of a geom_boxplot.

## Usage

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

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:<br>If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot`.<br>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.<br>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), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |
| geom, stat | Use to override the default connection between geom_quantile and stat_quantile. |
| quantiles | conditional quantiles of y to calculate and display |
| formula | formula relating y variables to x variables |
| method | Quantile regression method to use. Currently only supports rq. |
| method.args | List of additional arguments passed on to the modelling function defined by method. |

## Aesthetics

geom_quantileunderstands the following aesthetics (required aesthetics are in bold):

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

## 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 *Draw 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). geom_tile uses the center of the tile and its size (x, y, width, height). geom_raster is a high performance special case for when all the tiles are the same size.

### Usage

```
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:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>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. |
| 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), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |

## Aesthetics

geom_tileunderstands the following aesthetics (required aesthetics are in bold):

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

## 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(faithfuld, aes(waiting, eruptions)) +
 geom_raster(aes(fill = density))
```

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

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


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

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

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

---

`geom_ribbon` *Ribbons and area plots.*

---

## Description

For each continuous x value, `geom_interval` displays a y interval. `geom_area` is a special case of `geom_ribbon`, where the minimum of the range is fixed to 0.

## Usage

```
geom_ribbon(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", ..., na.rm = FALSE, show.legend = NA,
```

```
  inherit.aes = TRUE)

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

## Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by aes or aes_. If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |
| data | The data to be displayed in this layer. There are three options:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>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), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |

## 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.

## Aesthetics

geom_ribbonunderstands the following aesthetics (required aesthetics are in bold):

- **x**
- **ymax**

- **ymin**
- alpha
- colour
- fill
- linetype
- size

## See Also

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

## Examples

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

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

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

---

# geom_rug *Marginal rug plots.*

---

## Description

Marginal rug plots.

## Usage

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

## Arguments

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

| | |
|---|---|
| data | The data to be displayed in this layer. There are three options:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>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), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |

## Aesthetics

geom_rugunderstands the following aesthetics (required aesthetics are in bold):

- alpha
- colour
- linetype
- size

## Examples

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

geom_segment *Line segments and curves.*

## Description

geom_segment draws a straight line between points (x1, y1) and (x2, y2). geom_curve draws a curved line.

## Usage

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

geom_curve(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", ..., curvature = 0.5, angle = 90, ncp = 5,
  arrow = NULL, lineend = "butt", na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE)
```

## Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by aes or aes_. If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |
| data | The data to be displayed in this layer. There are three options:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>A function will be called with a single argument, the plot data. The return value must be a data.frame., and will be used as the layer data. |
| stat | The statistical transformation to use on the data for this layer, as a string. |
| position | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| ... | other arguments passed on to layer. These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat. |
| arrow | specification for arrow heads, as created by arrow() |
| lineend | Line end style (round, butt, square) |
| na.rm | If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |

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

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.

## Aesthetics

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

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

## See Also

geom_path and geom_line for multi- segment lines and paths.

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

## Examples

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

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

# 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` *Add a smoothed conditional mean.*

---

## 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

```
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:<br>If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot`.<br>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.<br>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, gam, loess, rlm. For datasets with n < 1000 default is loess. For datasets with 1000 or more observations defaults to gam, see gam for more details. |
| 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 |
| na.rm | If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |
| geom, stat | Use to override the default connection between geom_smooth and stat_smooth. |
| n | number of points to evaluate smoother at |
| span | Controls the amount of smoothing for the default loess smoother. Smaller numbers produce wigglier lines, larger numbers produce smoother lines. |
| fullrange | should the fit span the full range of the plot, or just the data |
| level | level of confidence interval to use (0.95 by default) |
| method.args | List of additional arguments passed on to the modelling function defined by 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_smoothunderstands the following aesthetics (required aesthetics are in bold):

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

### Computed variables

**y** predicted value

**ymin** lower pointwise confidence interval around the mean

**ymax** upper pointwise confidence interval around the mean

**se** standard error

### See Also

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

### Examples

```
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.
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_smooth(span = 0.3)

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

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

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


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.
ggplot(rpart::kyphosis, aes(Age, Kyphosis)) +
```

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

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

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

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

---

`geom_spoke` *A line segment parameterised by location, direction and distance.*

---

## Description

A line segment parameterised by location, direction and distance.

## Usage

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

## Arguments

| | |
|---|---|
| `mapping` | Set of aesthetic mappings created by `aes` or `aes_`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping. |
| `data` | The data to be displayed in this layer. There are three options:<br>If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot`.<br>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.<br>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), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |

## Aesthetics

geom_spokeunderstands the following aesthetics (required aesthetics are in bold):

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

## Examples

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

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

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

---

# geom_tallrect *ggplot2 geom with xmin and xmax aesthetics that covers the entire y range, useful for clickSelects background elements.*

---

## Description

ggplot2 geom with xmin and xmax aesthetics that covers the entire y range, useful for clickSelects background elements.

## Usage

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

## Arguments

| | |
|---|---|
| mapping | aesthetic mapping |
| data | data set |
| stat | statistic mapping, defaults to identity |
| position | position mapping, defaults to identity |
| ... | other arguments |
| na.rm | remove missing values? |
| show.legend | TRUE or FALSE |
| inherit.aes | TRUE or FALSE |

## Value

ggplot2 layer

## Examples

```
library(animint2)

## Example: 2 plots, 2 selectors, but only interacting with 1 plot.
data(breakpoints)
only.error <- subset(breakpoints$error,type=="E")
only.segments <- subset(only.error,bases.per.probe==bases.per.probe[1])
signal.colors <- c(estimate="#0adb0a",
                   latent="#0098ef")
breakpointError <-
  list(signal=ggplot()+
         geom_point(aes(position, signal),
                    showSelected="bases.per.probe",
                    data=breakpoints$signals)+
         geom_line(aes(position, signal), colour=signal.colors[["latent"]],
                   data=breakpoints$imprecision)+
         geom_segment(aes(first.base, mean, xend=last.base, yend=mean),
                      showSelected=c("segments", "bases.per.probe"),
                      colour=signal.colors[["estimate"]],
                      data=breakpoints$segments)+
         geom_vline(aes(xintercept=base),
                    showSelected=c("segments", "bases.per.probe"),
                    colour=signal.colors[["estimate"]],
                    linetype="dashed",
                    data=breakpoints$breaks),
       error=ggplot()+
         geom_vline(aes(xintercept=segments), clickSelects="segments",
                    data=only.segments, lwd=17, alpha=1/2)+
```

```
          geom_line(aes(segments, error, group=bases.per.probe),
                    clickSelects="bases.per.probe",
                    data=only.error, lwd=4))
animint2dir(breakpointError)
```

---

`geom_violin` *Violin plot.*

---

## Description

Violin plot.

## Usage

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

stat_ydensity(mapping = NULL, data = NULL, geom = "violin",
  position = "dodge", ..., bw = "nrd0", adjust = 1,
  kernel = "gaussian", trim = TRUE, scale = "area", na.rm = FALSE,
  show.legend = NA, inherit.aes = TRUE)
```

## Arguments

| | |
|---|---|
| `mapping` | Set of aesthetic mappings created by `aes` or `aes_`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping. |
| `data` | The data to be displayed in this layer. There are three options:<br>If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot`.<br>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.<br>A `function` will be called with a single argument, the plot data. The return value must be a `data.frame.`, and will be used as the layer data. |
| `position` | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| `...` | other arguments passed on to `layer`. These are often aesthetics, used to set an aesthetic to a fixed value, like `color = "red"` or `size = 3`. They may also be parameters to the paired geom/stat. |
| `draw_quantiles` | If `not(NULL)` (default), draw horizontal lines at the given quantiles of the density estimate. |

| | |
|---|---|
| trim | If TRUE (default), trim the tails of the violins to the range of the data. If FALSE, don't trim the tails. |
| scale | if "area" (default), all violins have the same area (before trimming the tails). If "count", areas are scaled proportionally to the number of observations. If "width", all violins have the same maximum width. |
| na.rm | If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |
| geom, stat | Use to override the default connection between geom_violin and stat_ydensity. |
| bw | the smoothing bandwidth to be used, see density for details |
| adjust | adjustment of the bandwidth, see density for details |
| kernel | kernel used for density estimation, see density for details |

## Aesthetics

geom_violinunderstands the following aesthetics (required aesthetics are in bold):

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

## Computed variables

**density** density estimate

**scaled** density estimate, scaled to maximum of 1

**count** density * number of points - probably useless for violin plots

**violinwidth** density scaled for the violin plot, according to area, counts or to a constant maximum width

**n** number of points

**width** width of violin bounding box

## References

Hintze, J. L., Nelson, R. D. (1998) Violin Plots: A Box Plot-Density Trace Synergism. The American Statistician 52, 181-184.

## See Also

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

## Examples

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


p + geom_violin() + geom_jitter(height = 0)
p + geom_violin() + coord_flip()

# 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")
}
```

---

`geom_widerect` *ggplot2 geom with ymin and ymax aesthetics that covers the entire x range, useful for clickSelects background elements.*

---

## Description

ggplot2 geom with ymin and ymax aesthetics that covers the entire x range, useful for clickSelects background elements.

## Usage

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

## Arguments

| | |
|---|---|
| `mapping` | aesthetic mapping |
| `data` | data set |
| `stat` | statistic mapping, defaults to identity |
| `position` | position mapping, defaults to identity |
| `...` | other arguments |
| `na.rm` | remove missing values? |
| `show.legend` | TRUE OR FALSE |
| `inherit.aes` | TRUE OR FALSE |

## Value

ggplot2 layer

## Examples

```
 ## Not run:
   source(system.file("examples/WorldBank.R", package = "animint"))

## End(Not run)
```

## getCommonChunk *Save the common columns for each tsv to one chunk*

### Description

Save the common columns for each tsv to one chunk

### Usage

```
getCommonChunk(built, chunk.vars, aes.list)
```

### Arguments

| | |
|---|---|
| built | data.frame of built data. |
| chunk.vars | which variables to chunk on. |
| aes.list | a character vector of aesthetics. |
| vars | character vector of chunk variable names to split on. |

### Value

a list of common and varied data to save, or NULL if there is no common data.

## getLayerName *Gives a unique name to each layer in* saveLayer

### Description

Gives a unique name to each layer in saveLayer

### Usage

```
getLayerName(L, geom_num, p.name)
```

### Arguments

| | |
|---|---|
| L | layer in saveLayer to be named |
| geom_num | the number of the layer to be saved |
| p.name | the name of the plot to which the layer belongs |

### Value

a unique name for the layer

## getLayerParams *Get all parameters for a layer*

### Description

Get all parameters for a layer

### Usage

```
getLayerParams(l)
```

### Arguments

| | |
|---|---|
| l | A single layer of the plot |

### Value

All parameters in the layer

## getLegend *Function to get legend information for each scale*

### Description

Function to get legend information for each scale

### Usage

```
getLegend(mb)
```

### Arguments

| | |
|---|---|
| mb | single entry from guides_merge() list of legend data |

### Value

list of legend information, NULL if guide=FALSE.

## getLegendList *Function to get legend information from ggplot*

### Description

Function to get legend information from ggplot

### Usage

```
getLegendList(plistextra)
```

### Arguments

| | |
|---|---|
| plistextra | output from ggplot_build(p) |

### Value

list containing information for each legend

## getUniqueAxisLabels *Get unique axis labels for the plot*

### Description

Get unique axis labels for the plot

### Usage

```
getUniqueAxisLabels(plot.meta)
```

### Arguments

| | |
|---|---|
| plot.meta | contains axis labels |

### Value

modified plot.meta with unique axis labels

## gganimintproto *Create a new gganimintproto object*

### Description

gganimintproto is inspired by the proto package, but it has some important differences. Notably, it cleanly supports cross-package inheritance, and has faster performance.

### Usage

```
gganimintproto(`_class` = NULL, `_inherit` = NULL, ...)

gganimintproto_parent(parent, self)
```

### Arguments

| | |
|---|---|
| `_class` | Class name to assign to the object. This is stored as the class attribute of the object. If `NULL` (the default), no class name will be added to the object. |
| `_inherit` | gganimintproto object to inherit from. If `NULL`, don't inherit from any object. |
| `...` | A list of members in the gganimintproto object. |
| `parent, self` | Access parent class `parent` of object `self`. |

### Calling gganimintproto methods

gganimintproto 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 gganimintproto 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 `gganimintproto_parent(Parent,self)`.

## ggplot *Create a new ggplot plot.*

### 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 suppled in each layer added to the plot. |
| mapping | Default list of aesthetic mappings to use for plot. If not specified, must be supplied in each layer added to the plot. |
| ... | Other arguments passed on to methods. Not currently used. |
| environment | If an variable defined in the aesthetic mapping is not found in the data, ggplot will look for it in this environment. It defaults to using the environment in which ggplot() is called. |

## Details

ggplot() is typically used to construct a plot incrementally, using the + operator to add layers to the existing ggplot object. This is advantageous in that the code is explicit about which layers are added and the order in which they are added. For complex graphics with multiple layers, initialization with ggplot is recommended.

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

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

# Declare the data frame and common aesthetics.
# The summary data frame ds is used to plot
# larger red points in a second geom_point() layer.
# If the data = argument is not specified, it uses the
# declared data frame from ggplot(); ditto for the aesthetics.
```

```
ggplot(df, aes(x = gp, y = 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(x = gp, y = y)) +
   geom_point(data = ds, aes(x = gp, y = mean),
                  colour = 'red', size = 3)
# Set up a skeleton ggplot object and add layers:
ggplot() +
  geom_point(data = df, aes(x = gp, y = y)) +
  geom_point(data = ds, aes(x = gp, y = mean),
                        colour = 'red', size = 3) +
  geom_errorbar(data = ds, aes(x = gp, y = mean,
                    ymin = mean - sd, ymax = mean + sd),
                    colour = 'red', width = 0.4)
```

---

`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 (function or any of the recognized extensions, e.g. `"pdf"`). By default, extracted from filename extension. `ggsave` currently recognises eps/ps, tex (pictex), pdf, jpeg, tiff, png, bmp, svg and wmf (windows only). |
| `path` | Path to save plot to (combined with filename). |
| `scale` | Multiplicative scaling factor. |
| `width, height` | Plot dimensions, defaults to size of current graphics device. |
| `units` | Units for width and height when specified explicitly (in, cm, or mm) |

| | |
|---|---|
| dpi | Resolution used for raster outputs. |
| 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

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

ggsave(file.path(tempdir(), "mtcars.pdf"))
ggsave(file.path(tempdir(), "mtcars.png"))

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

unlink(file.path(tempdir(), "mtcars.pdf"))
unlink(file.path(tempdir(), "mtcars.png"))

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

## End(Not run)
```

---

# ggtheme *ggplot2 themes*

---

## Description

Themes set the general aspect of the plot such as the colour of the background, gridlines, the size and colour of fonts.

## Usage

```
theme_grey(base_size = 11, base_family = "")

theme_gray(base_size = 11, base_family = "")

theme_bw(base_size = 12, base_family = "")

theme_linedraw(base_size = 12, base_family = "")

theme_light(base_size = 12, base_family = "")

theme_minimal(base_size = 12, base_family = "")
```

```
theme_classic(base_size = 12, base_family = "")

theme_dark(base_size = 12, base_family = "")

theme_void(base_size = 12, base_family = "")
```

## Arguments

| | |
|---|---|
| base_size | base font size |
| base_family | base font family |

## Details

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

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

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

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

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

theme_minimal A minimalistic theme with no background annotations.

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

theme_void A completely empty theme.

## Examples

```
p <- ggplot(mtcars) + geom_point(aes(x = wt, y = mpg,
     colour = factor(gear))) + facet_wrap(~am)

p
p + theme_gray()
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 in call of `scale_*` with argument `guide`, or in `guides`.

## Usage

```
guides(...)
```

## Arguments

| | |
|---|---|
| `...` | List of scale guide pairs |

## 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

p + theme(legend.position = "bottom", legend.box = "horizontal")

# 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[2].

## Usage

```
guide_colourbar(title = waiver(), title.position = NULL,
  title.theme = NULL, title.hjust = NULL, title.vjust = NULL,
  label = TRUE, label.position = NULL, label.theme = NULL,
  label.hjust = NULL, label.vjust = NULL, barwidth = NULL,
  barheight = NULL, nbin = 20, raster = TRUE, ticks = TRUE,
  draw.ulim = TRUE, draw.llim = TRUE, direction = NULL,
  default.unit = "line", reverse = FALSE, order = 0, ...)

guide_colorbar(title = waiver(), title.position = NULL,
  title.theme = NULL, title.hjust = NULL, title.vjust = NULL,
  label = TRUE, label.position = NULL, label.theme = NULL,
  label.hjust = NULL, label.vjust = NULL, barwidth = NULL,
```

[2] http://www.mathworks.com/help/techdoc/ref/colorbar.html

```
barheight = NULL, nbin = 20, raster = TRUE, ticks = TRUE,
draw.ulim = TRUE, draw.llim = TRUE, direction = NULL,
default.unit = "line", reverse = FALSE, order = 0, ...)
```

## Arguments

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

order — positive integer less 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_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

```
guide_legend(title = waiver(), title.position = NULL,
  title.theme = NULL, title.hjust = NULL, title.vjust = NULL,
  label = TRUE, label.position = NULL, label.theme = NULL,
  label.hjust = NULL, label.vjust = NULL, keywidth = NULL,
  keyheight = NULL, direction = NULL, default.unit = "line",
  override.aes = list(), nrow = NULL, ncol = NULL, byrow = FALSE,
  reverse = FALSE, order = 0, ...)
```

## Arguments

| | |
|---|---|
| `title` | A character string or expression indicating a title of guide. If `NULL`, the title is not shown. By default (`waiver`), the name of the scale object or the name specified in `labs` is used for the title. |
| `title.position` | A character string indicating the position of a title. One of "top" (default for a vertical guide), "bottom", "left" (default for a horizontal guide), or "right." |
| `title.theme` | A theme object for rendering the title text. Usually the object of `element_text` is expected. By default, the theme is specified by `legend.title` in `theme` or theme. |
| `title.hjust` | A number specifying horizontal justification of the title text. |

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

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

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

# Basic form
p1 + scale_fill_continuous(guide = "legend")
p1 + scale_fill_continuous(guide = guide_legend())

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

# Control styles

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

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

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

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

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

# Set aesthetic of legend key
```

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

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

# multiple row/col legends
df <- data.frame(x = 1:20, y = 1:20, color = letters[1:20])
p <- ggplot(df, aes(x, y)) +
  geom_point(aes(colour = color))
p + guides(col = guide_legend(nrow = 8))
p + guides(col = guide_legend(ncol = 8))
p + guides(col = guide_legend(nrow = 8, byrow = TRUE))
p + guides(col = guide_legend(ncol = 8, byrow = TRUE))

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

---

`hmisc` *Wrap up a selection of summary functions from Hmisc to make it easy to use with* `stat_summary`.

---

## Description

See the Hmisc documentation for details of their options.

## Usage

```
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. |

## See Also

`smean.cl.boot`, `smean.cl.normal`, `smean.sdl`, `smedian.hilow`

## intreg *Interval regression*

### Description

Learning model complexity using max-margin interval regression. We have observed several noisy piecewise constant signals, and we have weak labels about how many change-points occur in several regions. Max margin interval regression is an algorithm that uses this information to learn a penalty function for accurate change-point detection.

### Usage

```
data(intreg)
```

### Format

There are 7 related data.frames: signals contains the noisy piecewise constant signals, annotations contains the weak labels, segments and breaks contain the segmentation model, selection contains the penalty and cost information, intervals contains the target intervals of penalty values for each signal, and model describes the learned max margin interval regression model.

## is.gganimintproto *Is an object a gganimintproto object?*

### Description

Is an object a gganimintproto object?

### Usage

```
is.gganimintproto(x)
```

### Arguments

| | |
|---|---|
| x | An object to test. |

## is.rel *Reports whether x is a rel object*

### Description

Reports whether x is a rel object

### Usage

```
is.rel(x)
```

### Arguments

| | |
|---|---|
| x | An object to test |

## is.rgb *Check if character is an RGB hexadecimal color value*

### Description

Check if character is an RGB hexadecimal color value

### Usage

```
is.rgb(x)
```

### Arguments

| | |
|---|---|
| x | character |

### Value

True/False value

## is.theme *Reports whether x is a theme object*

### Description

Reports whether x is a theme object

### Usage

```
is.theme(x)
```

### Arguments

| | |
|---|---|
| x | An object to test |

## issueSelectorWarnings *Issue warnings for selectors*

### Description

Issue warnings for selectors

### Usage

```
issueSelectorWarnings(geoms, selector.aes, duration)
```

### Arguments

| | |
|---|---|
| geoms | geoms to check for warnings |
| selector.aes | selectors for each geom |
| duration | animation variable information to check for key value |

### Value

NULL

## knit_print.animint *Insert an interactive animation into an R markdown document using a customized print method.*

### Description

Insert an interactive animation into an R markdown document using a customized print method.

### Usage

```
knit_print.animint(x, options, ...)
```

### Arguments

| | |
|---|---|
| x | named list of ggplots and option lists to pass to animint2dir. |
| options | knitr options. |
| ... | placeholder. |

### Author(s)

Carson Sievert

### References

https://github.com/yihui/knitr/blob/master/vignettes/knit_print.Rmd

## labeller *Generic labeller function for facets*

### 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 `capitalize`.

## Value

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

## See Also

`as_labeller()`, labellers

## Examples

```
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"
)
## Source: http://en.wikipedia.org/wiki/Wikipedia:Conservation_status

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

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

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

# labeller() is especially useful to act as a global labeller. You
# can set it up once and use it on a range of different plots with
# different facet specifications.

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

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

---

`labellers` *Labeller functions*

---

## Description

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

## Usage

```
label_value(labels, multi_line = TRUE)

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

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

label_parsed(labels, multi_line = TRUE)

label_wrap_gen(width = 25, multi_line = TRUE)
```

## Arguments

| | |
|---|---|
| `labels` | Data frame of labels. Usually contains only one element, but facetting over multiple factors entails multiple label variables. |
| `multi_line` | Whether to display the labels of multiple factors on separate lines. |
| `sep` | String separating variables and values. |
| `width` | Maximum number of characters before wrapping the strip. |

## Details

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

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

## Writing New Labeller Functions

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

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

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

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

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

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

## See Also

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

## Examples

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

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


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

# Displaying only the values or both the values and variables
# depending on whether multiple factors are facetted over
p + facet_grid(am ~ vs+cyl, labeller = label_context)

# Interpreting the labels as plotmath expressions
p + facet_grid(. ~ cyl2)
p + facet_grid(. ~ cyl2, labeller = label_parsed)
p + facet_wrap(~vs + cyl2, labeller = label_parsed)
```

---

# label_bquote *Backquoted labeller*

---

## Description

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

## Usage

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

### Arguments

| | |
|---|---|
| rows | Backquoted labelling expression for rows. |
| cols | Backquoted labelling expression for columns. |
| default | Default labeller function for the rows or the columns when no plotmath expression is provided. |

### See Also

labellers, `labeller()`,

### Examples

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

---

## labs *Change axis labels, legend titles, plot title/subtitle and below-plot caption.*

---

### Description

Change axis labels, legend titles, plot title/subtitle and below-plot caption.

### Usage

```
labs(...)

xlab(label)

ylab(label)

ggtitle(label, subtitle = NULL)
```

### Arguments

| | |
|---|---|
| ... | a list of new names in the form aesthetic = "new name" |
| 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. |

### Examples

```
p <- ggplot(mtcars, aes(mpg, wt)) + geom_point()
p + labs(title = "New plot title")
p + labs(x = "New x label")
p + xlab("New x label")
p + ylab("New y label")
p + ggtitle("New plot title")

# Can add a subtitle to plots with either of the following
p + ggtitle("New plot title", subtitle = "A subtitle")
p + labs(title = "New plot title", subtitle = "A subtitle")

# Can add a plot caption underneath the whole plot (for sources, notes or
# copyright), similar to the \code{sub} parameter in base R, with the
# following
p + labs(caption = "(based on data from ...)")

# This should work independently of other functions that modify the
# the scale names
p + ylab("New y label") + ylim(2, 4)
p + ylim(2, 4) + ylab("New y label")

# The labs function also modifies legend labels
p <- ggplot(mtcars, aes(mpg, wt, colour = cyl)) + geom_point()
p + labs(colour = "Cylinders")

# Can also pass in a list, if that is more convenient
p + labs(list(title = "Title", subtitle = "Subtitle", x = "X", y = "Y"))
```

---

## last_plot *Retrieve the last plot to be modified or created.*

---

### Description

Retrieve the last plot to be modified or created.

### Usage

```
last_plot()
```

### See Also

ggsave

layer *Create a new layer*

## Description

A layer is a combination of data, stat and geom with a potential position adjustment. Usually layers are created using `geom_*` or `stat_*` calls but it can also be created directly using this function.

## Usage

```
layer(geom = NULL, stat = NULL, data = NULL, mapping = NULL,
  position = NULL, params = list(), inherit.aes = TRUE,
  subset = NULL, show.legend = NA)
```

## Arguments

| | |
|---|---|
| `geom` | The geometric object to use display the data |
| `stat` | The statistical transformation to use on the data for this layer, as a string. |
| `data` | The data to be displayed in this layer. There are three options:<br>If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot`.<br>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.<br>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. |
| `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. |
| `position` | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| `params` | Additional parameters to the `geom` and `stat`. |
| `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`. |
| `subset` | DEPRECATED. An older way of subsetting the dataset used in a layer. |
| `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. |

## Examples

```
# geom calls are just a short cut for layer
ggplot(mpg, aes(displ, hwy)) + geom_point()
# shortcut for
```

```
ggplot(mpg, aes(displ, hwy)) +
  layer(geom = "point", stat = "identity", position = "identity",
    params = list(na.rm = FALSE)
  )

# use a function as data to plot a subset of global data
ggplot(mpg, aes(displ, hwy)) +
  layer(geom = "point", stat = "identity", position = "identity",
    data = head, params = list(na.rm = FALSE)
  )
```

---

`lims` *Convenience functions to set the axis limits.*

---

## Description

Observations not in this range will be dropped completely and not passed to any other layers. If a NA value is substituted for one of the limits that limit is automatically calculated.

## Usage

```
lims(...)

xlim(...)

ylim(...)
```

## Arguments

| | |
|---|---|
| `...` | If numeric, will create a continuous scale, if factor or character, will create a discrete scale. For `lims`, every argument must be named. |

## See Also

For changing x or y axis limits **without** dropping data observations, see `coord_cartesian`.

## Examples

```
# xlim
xlim(15, 20)
xlim(20, 15)
xlim(c(10, 20))
xlim("a", "b", "c")

ggplot(mtcars, aes(mpg, wt)) +
  geom_point() +
  xlim(15, 20)
# with automatic lower limit
```

```
ggplot(mtcars, aes(mpg, wt)) +
  geom_point() +
  xlim(NA, 20)

# Change both xlim and ylim
ggplot(mtcars, aes(mpg, wt)) +
  geom_point() +
  lims(x = c(10, 20), y = c(3, 5))
```

---

`luv_colours` `colors()` *in Luv space.*

---

## Description

All built-in `colors()` translated into Luv colour space.

## Usage

```
luv_colours
```

## Format

A data frame with 657 observations and 4 variables:

- L,u,vPosition in Luv colour space
- colColour name

---

`make_bar` *Convenience function for an interactive bar that might otherwise be created using stat_summary(geom="bar").*

---

## Description

Convenience function for an interactive bar that might otherwise be created using stat_summary(geom="bar").

## Usage

```
make_bar(data, x.name, alpha = 1)
```

## Arguments

| | |
|---|---|
| `data` | data.frame to analyze for unique x.name values. |
| `x.name` | variable to be used for x, clickSelects. |
| `alpha` | transparency of selected bar, default 1. |

## Value

a geom_bar layer.

## Author(s)

Toby Dylan Hocking

---

`make_tallrect` *Make a clickSelects geom_tallrect that completely tiles the x range. This makes it easy to construct tallrects for the common case of selecting a particular x value.*

---

## Description

Make a clickSelects geom_tallrect that completely tiles the x range. This makes it easy to construct tallrects for the common case of selecting a particular x value.

## Usage

```
make_tallrect(data, x.name, even = FALSE, alpha = 1/2, ...)
```

## Arguments

| | |
|---|---|
| `data` | data.frame to analyze for unique x.name values. |
| `x.name` | variable to be used for x, clickSelects. |
| `even` | Logical parameter, should tallrects be of even width? |
| `alpha` | transparency of a selected tallrect, default 1/2. |
| `...` | passed to geom_tallrect. |

## Value

a geom_tallrect layer.

## Author(s)

Toby Dylan Hocking

`make_tallrect_or_widerect` *Make a clickSelects geom_widerect or geom_tallrect that completely tiles the x or y range. This function is used internally by make_tallrect or make_widerect, which are more user-friendly.*

## Description

Make a clickSelects geom_widerect or geom_tallrect that completely tiles the x or y range. This function is used internally by make_tallrect or make_widerect, which are more user-friendly.

## Usage

```
make_tallrect_or_widerect(aes.prefix, geom_xrect, data, var.name,
  even = FALSE, alpha = 0.5, ..., data.fun = identity)
```

## Arguments

| | |
|---|---|
| `aes.prefix` | "x" or "y" |
| `geom_xrect` | geom_tallrect or geom_widerect |
| `data` | data.frame to analyze for unique var.name values. |
| `var.name` | variable to be used for clickSelects |
| `even` | Logical parameter, should xrects be of even width? |
| `alpha` | transparency of a selected xrect, default 1/2. |
| `...` | passed to geom_xrect |
| `data.fun` | called on data passed to geom_xrect(aes(..), data.fun(df)) this is useful in facetted plots, for adding columns to the data.frame, if you want that geom in only one panel. |

## Value

a geom_xrect layer

## Author(s)

Toby Dylan Hocking

## make_text *Convenvience function for a showSelected plot label.*

### Description

Convenvience function for a showSelected plot label.

### Usage

```
make_text(data, x, y, label.var, format = NULL)
```

### Arguments

| | |
|---|---|
| data | data.frame of relevant data |
| x | x coordinate of label position |
| y | y coordinate of label position |
| label.var | variable matching showSelected, used to obtain label value |
| format | String format for label. Use %d, %f, etc. to insert relevant label.var value. |

### Value

a geom_text layer.

### Author(s)

Toby Dylan Hocking

## make_widerect *Make a clickSelects geom_widerect that completely tiles the y range. This makes it easy to construct widerects for the common case of selecting a particular y value.*

### Description

Make a clickSelects geom_widerect that completely tiles the y range. This makes it easy to construct widerects for the common case of selecting a particular y value.

### Usage

```
make_widerect(data, y.name, even = FALSE, alpha = 0.5, ...)
```

### Arguments

| | |
|---|---|
| data | data.frame to analyze for unique y.name values. |
| y.name | variable to be used for y, clickSelects. |
| even | Logical parameter, should widerects be of even width? |
| alpha | transparency of a selected widerect, default 1/2. |
| ... | passed to geom_widerect. |

### Value

a geom_widerect layer.

### Author(s)

Toby Dylan Hocking

---

## map_data *Create a data frame of map data.*

---

### Description

Create a data frame of map data.

### Usage

```
map_data(map, region = ".", exact = FALSE, ...)
```

### Arguments

| | |
|---|---|
| map | name of map provided by the **maps** package. These include county, france, italy, nz, state, usa, world, world2. |
| region | name of subregions to include. Defaults to . which includes all subregion. See documentation for map for more details. |
| exact | should the region be treated as a regular expression (FALSE) or as a fixed string (TRUE). |
| ... | all other arguments passed on to map |

### Examples

```
if (require("maps")) {
states <- map_data("state")
arrests <- USArrests
names(arrests) <- tolower(names(arrests))
arrests$region <- tolower(rownames(USArrests))

choro <- merge(states, arrests, sort = FALSE, by = "region")
choro <- choro[order(choro$order), ]
```

```
ggplot(choro, aes(long, lat)) +
  geom_polygon(aes(group = group, fill = assault)) +
  coord_map("albers",  at0 = 45.5, lat1 = 29.5)

ggplot(choro, aes(long, lat)) +
  geom_polygon(aes(group = group, fill = assault / murder)) +
  coord_map("albers",  at0 = 45.5, lat1 = 29.5)
}
```

---

`margin` *Define margins.*

---

## Description

This is a convenient function that creates a grid unit object of the correct length to use for setting margins.

## Usage

```
margin(t = 0, r = 0, b = 0, l = 0, unit = "pt")
```

## 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. |

## Examples

```
margin(4)
margin(4, 2)
margin(4, 3, 2, 1)
```

---

`mean_se` *Calculate mean and standard errors on either side.*

---

## Description

Calculate mean and standard errors on either side.

## Usage

```
mean_se(x, mult = 1)
```

### Arguments

| | |
|---|---|
| x | numeric vector |
| mult | number of multiples of standard error |

### See Also

for use with stat_summary

---

## merge_recurse *Merge a list of data frames.*

---

### Description

Merge a list of data frames.

### Usage

```
merge_recurse(dfs)
```

### Arguments

| | |
|---|---|
| dfs | list of data frames |

### Value

data frame

---

## midwest *Midwest demographics.*

---

### Description

Demographic information of midwest counties

### Usage

```
midwest
```

**Format**

A data frame with 437 rows and 28 variables

- PID
- county
- state
- area
- poptotal. Total population
- popdensity. Population density
- popwhite. Number of whites.
- popblack. Number of blacks.
- popamerindian. Number of American Indians.
- popasian. Number of Asians.
- popother. Number of other races.
- percwhite. Percent white.
- percblack. Percent black.
- percamerindan. Percent American Indian.
- percasian. Percent Asian.
- percother. Percent other races.
- popadults. Number of adults.
- perchsd.
- percollege. Percent college educated.
- percprof. Percent profession.
- poppovertyknown.
- percpovertyknown
- percbelowpoverty
- percchildbelowpovert
- percadultpoverty
- percelderlypoverty
- inmetro. In a metro area.
- category'

## mpg — *Fuel economy data from 1999 and 2008 for 38 popular models of car*

### Description

This dataset contains a subset of the fuel economy data that the EPA makes available on `http://fueleconomy.gov`. 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.
- displ. engine displacement, in litres
- year.
- cyl. number of cylinders
- trans. type of transmission
- drv. f = front-wheel drive, r = rear wheel drive, 4 = 4wd
- cty. city miles per gallon
- hwy. highway miles per gallon
- fl.
- class.

## msleep — *An updated and expanded version of the mammals sleep dataset.*

### Description

This is an updated and expanded version of the mammals sleep dataset. Updated sleep times and weights were taken from V. M. Savage and G. B. West. A quantitative, theoretical framework for understanding mammalian sleep. Proceedings of the National Academy of Sciences, 104 (3):1051-1056, 2007.

### Usage

```
msleep
```

### Format

A data frame with 83 rows and 11 variables

- name. common name
- genus.
- vore. carnivore, omnivore or herbivore?
- order.
- conservation. the conservation status of the animal
- sleep_total. total amount of sleep, in hours
- sleep_rem. rem sleep, in hours
- sleep_cycle. length of sleep cycle, in hours
- awake. amount of time spent awake, in hours
- brainwt. brain weight in kilograms
- bodywt. body weight in kilograms

### Details

Additional variables order, conservation status and vore were added from wikipedia.

---

## `newEnvironment` *Environment to store meta data*

---

### Description

Get a new environment to store meta-data. Used to alter state in the lower-level functions

### Usage

```
newEnvironment()
```

### Value

A new environment to store meta data

## parsePlot *Convert a ggplot to a list.*

### Description

Convert a ggplot to a list.

### Usage

```
parsePlot(meta, plot, plot.name)
```

### Arguments

| | |
|---|---|
| meta | environment with previously calculated plot data, and a new plot to parse, already stored in plot and plot.name. |
| plot | ggplot list object |
| plot.name | name of plot |

### Value

nothing, info is stored in meta.

## position_dodge *Adjust position by dodging overlaps to the side.*

### Description

Adjust position by dodging overlaps to the side.

### Usage

```
position_dodge(width = NULL)
```

### Arguments

| | |
|---|---|
| width | Dodging width, when different to the width of the individual elements. This is useful when you want to align narrow geoms with wider geoms. See the examples for a use case. |

### See Also

Other position adjustments: position_fill, position_identity, position_jitterdodge, position_jitter, position_nudge

### Examples

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

ggplot(diamonds, aes(price, fill = cut)) +
  geom_histogram(position="dodge")
# see ?geom_boxplot and ?geom_bar for more examples

# 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_bar(
    stat = "identity", 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")
# You need to explicitly specify the width for dodging
p + geom_linerange(aes(ymin = y-1, ymax = y+1),
  position = position_dodge(width = 0.9))

# Similarly with error 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 = 0.90))
```

---

## position_fill *Stack overlapping objects on top of one another.*

---

### Description

position_fill additionally standardises each stack to have unit height.

### Usage

```
position_fill()

position_stack()
```

### 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 is the default behaviour for most area plots:
ggplot(mtcars, aes(factor(cyl), fill = factor(vs))) + geom_bar()
# Fill makes it easier to compare proportions
ggplot(mtcars, aes(factor(cyl), fill = factor(vs))) +
  geom_bar(position = "fill")

# To change stacking order, use factor() to change order of levels
mtcars$vs <- factor(mtcars$vs, levels = c(1,0))
ggplot(mtcars, aes(factor(cyl), fill = factor(vs))) + geom_bar()

ggplot(diamonds, aes(price, fill = cut)) +
  geom_histogram(binwidth = 500)
# When used with a histogram, position_fill creates a conditional density
# estimate
ggplot(diamonds, aes(price, fill = cut)) +
  geom_histogram(binwidth = 500, position = "fill")

# Stacking is also useful for time series
data.set <- 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)
)

ggplot(data.set, aes(Time, Value)) + geom_area(aes(fill = Type))

# If you want to stack lines, you need to say so:
ggplot(data.set, aes(Time, Value)) + geom_line(aes(colour = Type))
ggplot(data.set, aes(Time, Value)) +
  geom_line(position = "stack", aes(colour = Type))

# But realise that this makes it *much* harder to compare individual
# trends
```

---

## position_identity *Don't adjust position*

---

### Description

Don't adjust position

### Usage

```
position_identity()
```

### See Also

Other position adjustments: position_dodge, position_fill, position_jitterdodge, position_jitter, position_nudge

position_jitter *Jitter points to avoid overplotting.*

## Description

Jitter points to avoid overplotting.

## Usage

```
position_jitter(width = NULL, height = NULL)
```

## Arguments

width, height

Amount of vertical and horizontal jitter. The jitter is added in both positive and negative directions, so the total spread is twice the value specified here.

If omitted, defaults to 40% of the resolution of the data: this means the jitter values will occupy 80% of the implied bins. Categorical data is aligned on the integers, so a width or height of 0.5 will spread the data so it's not possible to see the distinction between the categories.

## See Also

Other position adjustments: position_dodge, position_fill, position_identity, position_jitterdodge, position_nudge

## Examples

```
ggplot(mtcars, aes(am, vs)) + geom_point()

# Default amount of jittering will generally be too much for
# small datasets:
ggplot(mtcars, aes(am, vs)) + geom_jitter()

# Two ways to override
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))

# The default works better for large datasets, where it will
# take up as much space as a boxplot or a bar
ggplot(mpg, aes(class, hwy)) +
  geom_jitter() +
  geom_boxplot()
```

## position_jitterdodge *Adjust position by simultaneously dodging and jittering*

### 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

```
position_jitterdodge(jitter.width = NULL, jitter.height = 0,
  dodge.width = 0.75)
```

### Arguments

| | |
|---|---|
| `jitter.width` | degree of jitter in x direction. Defaults to 40% of the resolution of the data. |
| `jitter.height` | degree of jitter in y direction. Defaults to 0. |
| `dodge.width` | the amount to dodge in the x direction. Defaults to 0.75, the default `position_dodge()` width. |

### See Also

Other position adjustments: `position_dodge`, `position_fill`, `position_identity`, `position_jitter`, `position_nudge`

### 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.*

### Description

This is useful if you want to nudge labels a little ways from their points.

### 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_fill, position_identity, position_jitterdodge, position_jitter

### 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))
```

---

## 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
```

### Format

A data frame with 11 rows and 4 variables

## print.animint *print animint*

### Description

Print animint by rendering to local directory.

### Usage

```
## S3 method for class 'animint'
print(x, ...)
```

### Arguments

| | |
|---|---|
| x | List of ggplots and options. In particular the out.dir option is passed along to animint2dir. |
| ... | passed to animint2dir |

### Value

same as animint2dir

### Author(s)

Toby Dylan Hocking

## print.gganimintplot *Draw plot on current graphics device.*

### Description

Draw plot on current graphics device.

### Usage

```
## S3 method for class 'gganimintplot'
print(x, newpage = is.null(vp), vp = NULL, ...)

## S3 method for class 'gganimintplot'
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.

---

## print.gganimintproto

*Print a gganimintproto object*

---

### Description

If a gganimintproto 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

```
## S3 method for class 'gganimintproto'
print(x, ..., flat = TRUE)
```

### Arguments

| | |
|---|---|
| x | A gganimintproto object to print. |
| ... | If the gganimintproto 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. |

## pt.to.lines *Convert* pt *value to* lines

### Description

Convert pt value to lines

### Usage

```
pt.to.lines(pt_value)
```

### Arguments

| | |
|---|---|
| pt_value | Value in pt to be converted to lines |

### Value

Value in lines

### Note

Does NOT work if input is not in pt. Input is returned as is.

## qplot *Quick plot*

### Description

qplot is the basic plotting function in the ggplot2 package, 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.

### Usage

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

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

## 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")
```

---

## rel *Relative sizing for theme elements*

---

### Description

Relative sizing for theme elements

### Usage

```
rel(x)
```

### Arguments

| | |
|---|---|
| x | A number representing the relative size |

### Examples

```
df <- data.frame(x = 1:3, y = 1:3)
ggplot(df, aes(x, y)) +
  geom_point() +
  theme(axis.title.x = element_text(size = rel(2.5)))
```

---

## renderAnimint *Create an animint output element*

---

### Description

Shiny server output function customized for animint plots (similar to `shiny::plotOutput` and friends).

### Usage

```
renderAnimint(expr, env = parent.frame(), quoted = FALSE)
```

### Arguments

| | |
|---|---|
| expr | An expression that creates a list of ggplot objects. |
| env | The environment in which to evaluate `expr`. |
| quoted | Is expr a quoted expression (with `quote()`)? This is useful if you want to save an expression in a variable. |

### See Also

http://shiny.rstudio.com/articles/building-outputs.html

---

## resolution *Compute the "resolution" of a data vector.*

---

### Description

The resolution is is the smallest non-zero distance between adjacent values. If there is only one unique value, then the resolution is defined to be one.

### Usage

```
resolution(x, zero = TRUE)
```

### Arguments

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

### Details

If x is an integer vector, then it is assumed to represent a discrete variable, and the resolution is 1.

### Examples

```
resolution(1:10)
resolution((1:10) - 0.5)
resolution((1:10) - 0.5, FALSE)
resolution(c(1,2, 10, 20, 50))
resolution(as.integer(c(1, 10, 20, 50)))  # Returns 1
```

---

## saveChunks *Split data set into chunks and save them to separate files.*

---

### Description

Split data set into chunks and save them to separate files.

### Usage

```
saveChunks(x, meta)
```

### Arguments

| | |
|---|---|
| x | data.frame. |
| meta | environment. |

## Value

recursive list of chunk file names.

## Author(s)

Toby Dylan Hocking

---

`saveLayer` *Save a layer to disk, save and return meta-data.*

---

## Description

Save a layer to disk, save and return meta-data.

## Usage

```
saveLayer(l, d, meta, layer_name, ggplot, built, AnimationInfo)
```

## Arguments

| | |
|---|---|
| `l` | one layer of the ggplot object. |
| `d` | one layer of calculated data from ggplot_build(p). |
| `meta` | environment of meta-data. |
| `layer_name` | name of layer |
| `ggplot` | ggplot |
| `built` | built list |
| `AnimationInfo` | animation list ID number starting from 1 |

## Value

list representing a layer, with corresponding aesthetics, ranges, and groups.

## scale_alpha *Alpha scales.*

### Description

scale_alpha is an alias for scale_alpha_continuous since that is the most common use of alpha, and it saves a bit of typing.

### Usage

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

scale_alpha_continuous(..., range = c(0.1, 1))

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

### Arguments

| | |
|---|---|
| ... | Other arguments passed on to continuous_scale or discrete_scale as appropriate, to control name, limits, breaks, labels and so forth. |
| range | range of output alpha values. Should lie between 0 and 1. |

### Examples

```
(p <- ggplot(mtcars, aes(mpg, cyl)) +
  geom_point(aes(alpha = cyl)))
p + scale_alpha("cylinders")
p + scale_alpha("number\nof\ncylinders")

p + scale_alpha(range = c(0.4, 0.8))

(p <- ggplot(mtcars, aes(mpg, cyl)) +
  geom_point(aes(alpha = factor(cyl))))
p + scale_alpha_discrete(range = c(0.4, 0.8))
```

## scale_colour_brewer *Sequential, diverging and qualitative colour scales from colorbrewer.org*

## Description

ColorBrewer provides sequential, diverging and qualitative colour schemes which are particularly suited and tested to display discrete values (levels of a factor) on a map. ggplot2 can use those colours in discrete scales. It also allows to smoothly interpolate 6 colours from any palette to a continuous scale (6 colours per palette gives nice gradients; more results in more saturated colours which do not look as good). However, the original colour schemes (particularly the qualitative ones) were not intended for this and the perceptual result is left to the appreciation of the user. See `http://colorbrewer2.org` for more information.

## Usage

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

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

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

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

## Arguments

| | |
|---|---|
| `...` | Other arguments passed on to `discrete_scale` to control name, limits, breaks, labels and so forth. |
| `type` | One of seq (sequential), div (diverging) or qual (qualitative) |
| `palette` | If a string, will use that named palette. If a number, will index into the list of palettes of appropriate `type` |
| `direction` | Sets the order of colors in the scale. If 1, the default, colors are as output by `brewer.pal`. If -1, the order of colors is reversed. |
| `values` | if colours should not be evenly positioned along the gradient this vector gives the position (between 0 and 1) for each colour in the `colours` vector. See `rescale` for a convience function to map an arbitrary range to between 0 and 1. |
| `space` | colour space in which to calculate gradient. Must be "Lab" - other values are deprecated. |
| `na.value` | Colour to use for missing values |
| `guide` | Type of legend. Use `"colourbar"` for continuous colour bar, or `"legend"` for discrete colour legend. |

## 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

## See Also

Other colour scales: `scale_colour_gradient`, `scale_colour_grey`, `scale_colour_hue`

## Examples

```
dsamp <- diamonds[sample(nrow(diamonds), 1000), ]
(d <- ggplot(dsamp, aes(carat, price)) +
  geom_point(aes(colour = clarity)))

# Change scale label
d + scale_colour_brewer()
d + scale_colour_brewer("Diamond\nclarity")

# Select brewer palette to use, see ?scales::brewer_pal for more details
d + scale_colour_brewer(palette = "Greens")
d + scale_colour_brewer(palette = "Set1")


# scale_fill_brewer works just the same as
# scale_colour_brewer but for fill colours
p <- ggplot(diamonds, aes(x = price, fill = cut)) +
  geom_histogram(position = "dodge", binwidth = 1000)
p + scale_fill_brewer()
# the order of colour can be reversed
p + scale_fill_brewer(direction = -1)
# the brewer scales look better on a darker background
p + scale_fill_brewer(direction = -1) + theme_dark()


# Use distiller variant with continous data
v <- ggplot(faithfuld) +
  geom_tile(aes(waiting, eruptions, fill = density))
v
v + scale_fill_distiller()
v + scale_fill_distiller(palette = "Spectral")
```

---

# scale_colour_gradient

*Smooth gradient between two colours*

---

## Description

`scale_*_gradient` creates a two colour gradient (low-high), `scale_*_gradient2` creates a diverging colour gradient (low-mid-high), `scale_*_gradientn` creats a n-colour gradient.

## Usage

```
scale_colour_gradient(..., low = "#132B43", high = "#56B1F7",
  space = "Lab", na.value = "grey50", guide = "colourbar")

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

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

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

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

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

## Arguments

| | |
|---|---|
| `...` | Other arguments passed on to `discrete_scale` to control name, limits, breaks, labels and so forth. |
| `low, high` | Colours for low and high ends of the gradient. |
| `space` | colour space in which to calculate gradient. Must be "Lab" - other values are deprecated. |
| `na.value` | Colour to use for missing values |
| `guide` | Type of legend. Use `"colourbar"` for continuous colour bar, or `"legend"` for discrete colour legend. |
| `mid` | colour for mid point |
| `midpoint` | The midpoint (in data value) of the diverging scale. Defaults to 0. |
| `colours, colors` | Vector of colours to use for n-colour gradient. |
| `values` | if colours should not be evenly positioned along the gradient this vector gives the position (between 0 and 1) for each colour in the `colours` vector. See `rescale` for a convience function to map an arbitrary range to between 0 and 1. |

## Details

Default colours are generated with **munsell** and `mnsl(c("2.5PB 2/4","2.5PB 7/10")`. Generally, for continuous colour scales you want to keep hue constant, but vary chroma and luminance. The **munsell** package makes this easy to do using the Munsell colour system.

## See Also

seq_gradient_pal for details on underlying palette

Other colour scales: scale_colour_brewer, scale_colour_grey, scale_colour_hue

## Examples

```
df <- data.frame(
  x = runif(100),
  y = runif(100),
  z1 = rnorm(100),
  z2 = abs(rnorm(100))
)

# Default colour scale colours from light blue to dark blue
ggplot(df, aes(x, y)) +
  geom_point(aes(colour = z2))

# For diverging colour scales use gradient2
ggplot(df, aes(x, y)) +
  geom_point(aes(colour = z1)) +
  scale_colour_gradient2()

# Use your own colour scale with gradientn
ggplot(df, aes(x, y)) +
  geom_point(aes(colour = z1)) +
  scale_colour_gradientn(colours = terrain.colors(10))

# Equivalent fill scales do the same job for the fill aesthetic
ggplot(faithfuld, 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 scale.*

---

## Description

Based on gray.colors

### Usage

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

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

### Arguments

| | |
|---|---|
| ... | Other arguments passed on to discrete_scale to control name, limits, breaks, labels and so forth. |
| start | gray value at low end of palette |
| end | gray value at high end of palette |
| na.value | Colour to use for missing values |

### See Also

Other colour scales: scale_colour_brewer, scale_colour_gradient, scale_colour_hue

### 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 *Qualitative colour scale with evenly spaced hues.*

---

### Description

Qualitative colour scale with evenly spaced hues.

### Usage

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

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

## Arguments

| | |
|---|---|
| ... | Other arguments passed on to discrete_scale to control name, limits, breaks, labels and so forth. |
| h | range of hues to use, in [0, 360] |
| c | chroma (intensity of colour), maximum value varies depending on combination of hue and luminance. |
| l | luminance (lightness), in [0, 100] |
| h.start | hue to start at |
| direction | direction to travel around the colour wheel, 1 = clockwise, -1 = counter-clockwise |
| na.value | Colour to use for missing values |

## See Also

Other colour scales: scale_colour_brewer, scale_colour_gradient, scale_colour_grey

## Examples

```
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))
d + scale_colour_hue(h = c(90, 180))
d + scale_colour_hue(h = c(180, 270))
d + scale_colour_hue(h = c(270, 360))

# Vary opacity
# (only works with pdf, quartz and cairo devices)
d <- ggplot(dsamp, aes(carat, price, colour = clarity))
d + geom_point(alpha = 0.9)
d + geom_point(alpha = 0.5)
d + geom_point(alpha = 0.2)

# Colour of missing values is controlled with na.value:
miss <- factor(sample(c(NA, 1:5), nrow(mtcars), replace = TRUE))
ggplot(mtcars, aes(mpg, wt)) + geom_point(aes(colour = miss))
ggplot(mtcars, aes(mpg, wt)) +
```

```
  geom_point(aes(colour = miss)) +
  scale_colour_hue(na.value = "black")
```

---

`scale_continuous` *Continuous position scales (x & y).*

---

## Description

`scale_x_continuous` and `scale_y_continuous` are the key functions. The others, `scale_x_log10`, `scale_y_sqrt` etc, are aliases that set the `trans` argument to commonly used transformations.

## 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")

scale_y_continuous(name = waiver(), breaks = waiver(),
  minor_breaks = waiver(), labels = waiver(), limits = NULL,
  expand = waiver(), oob = censor, na.value = NA_real_,
  trans = "identity")

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 `NULL`, the default, the name of the scale is taken from the first mapping used for that aesthetic. |
| `breaks` | One of:<br>• `NULL` for no breaks<br>• `waiver()` for the default breaks computed by the transformation object<br>• A numeric vector of positions<br>• A function that takes the limits as input and returns breaks as output |
| `minor_breaks` | One of: |

- NULL for no minor breaks
- waiver() for the default breaks (one minor break between each major break)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks.

labels — One of:

- NULL for no labels
- waiver() for the default labels computed by the transformation object
- A character vector giving labels (must be same length as breaks)
- A function that takes the breaks as input and returns labels as output

limits — A numeric vector of length two providing limits of the scale. Use NA to refer to the existing minimum or maximum.

expand — A numeric vector of length two giving multiplicative and additive expansion constants. These constants ensure that the data is placed some distance away from the axes. The defaults are c(0.05,0) for continuous variables, and c(0,0.6) for discrete variables.

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

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

trans — Either the name of a transformation object, or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "exp", "identity", "log", "log10", "log1p", "log2", "logit", "probability", "probit", "reciprocal", "reverse" and "sqrt".

A transformation object bundles together a transform, 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. boxcox_trans. You can create your own transformation with trans_new.

... — Other arguments passed on to scale_(x|y)_continuous

## See Also

scale_date for date/time position scales.

## Examples

```
if (require(ggplot2movies)) {
m <- ggplot(subset(movies, votes > 1000), aes(rating, votes)) +
  geom_point(na.rm = TRUE)
m

# Manipulating the default position scales lets you:

#  * change the axis labels
m + scale_y_continuous("number of votes")
m + scale_y_continuous(quote(votes ^ alpha))
```

```
#  * modify the axis limits
m + scale_y_continuous(limits = c(0, 5000))
m + scale_y_continuous(limits = c(1000, 10000))
m + scale_x_continuous(limits = c(7, 8))

# you can also use the short hand functions xlim and ylim
m + ylim(0, 5000)
m + ylim(1000, 10000)
m + xlim(7, 8)

#  * choose where the ticks appear
m + scale_x_continuous(breaks = 1:10)
m + scale_x_continuous(breaks = c(1,3,7,9))

#  * manually label the ticks
m + scale_x_continuous(breaks = c(2,5,8), labels = c("two", "five", "eight"))
m + scale_x_continuous(breaks = c(2,5,8), labels = c("horrible", "ok", "awesome"))
m + scale_x_continuous(breaks = c(2,5,8), labels = expression(Alpha, Beta, Omega))

# There are a few built in transformation that you can use:
m + scale_y_log10()
m + scale_y_sqrt()
m + scale_y_reverse()
# You can also create your own and supply them to the trans argument.
# See ?scales::trans_new

# You can control the formatting of the labels with the formatter
# argument.  Some common formats are built into the scales package:
df <- data.frame(
  x = rnorm(10) * 100000,
  y = seq(0, 1, length.out = 10)
)
p <- ggplot(df, aes(x, y)) + geom_point()
p + scale_y_continuous(labels = scales::percent)
p + scale_y_continuous(labels = scales::dollar)
p + scale_x_continuous(labels = scales::comma)

# Other shortcut functions
ggplot(movies, aes(rating, votes)) +
  geom_point() +
  ylim(1e4, 5e4)
#   * axis labels
ggplot(movies, aes(rating, votes)) +
  geom_point() +
  labs(x = "My x axis", y = "My y axis")
#   * log scaling
ggplot(movies, aes(rating, votes)) +
  geom_point() +
  scale_x_log10() +
  scale_y_log10()
}
```

scale_date *Position scale, date & date times*

## Description

Use `scale_*_date` with `Date` variables, and `scale_*_datetime` with `POSIXct` variables.

## 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())

scale_y_date(name = waiver(), breaks = waiver(),
  date_breaks = waiver(), labels = waiver(), date_labels = waiver(),
  minor_breaks = waiver(), date_minor_breaks = waiver(),
  limits = NULL, expand = waiver())

scale_x_datetime(name = waiver(), breaks = waiver(),
  date_breaks = waiver(), labels = waiver(), date_labels = waiver(),
  minor_breaks = waiver(), date_minor_breaks = waiver(),
  limits = NULL, expand = waiver())

scale_y_datetime(name = waiver(), breaks = waiver(),
  date_breaks = waiver(), labels = waiver(), date_labels = waiver(),
  minor_breaks = waiver(), date_minor_breaks = waiver(),
  limits = NULL, expand = waiver())
```

## Arguments

| | |
|---|---|
| `name` | The name of the scale. Used as axis or legend title. If `NULL`, the default, the name of the scale is taken from the first mapping used for that aesthetic. |
| `breaks` | One of:<br>• `NULL` for no breaks<br>• `waiver()` for the default breaks computed by the transformation object<br>• A numeric vector of positions<br>• 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:<br>• `NULL` for no labels<br>• `waiver()` for the default labels computed by the transformation object<br>• A character vector giving labels (must be same length as `breaks`)<br>• A function that takes the breaks as input and returns labels as output |

`date_labels` A string giving the formatting specification for the labels. Codes are defined in `strftime`. If both `labels` and `date_labels` are specified, `date_labels` wins.

`minor_breaks` One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (one minor break between each major break)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks.

`date_minor_breaks`
A string giving the distance between minor breaks like "2 weeks", or "10 years". If both `minor_breaks` and `date_minor_breaks` are specified, `date_minor_breaks` wins.

`limits` A numeric vector of length two providing limits of the scale. Use `NA` to refer to the existing minimum or maximum.

`expand` A numeric vector of length two giving multiplicative and additive expansion constants. These constants ensure that the data is placed some distance away from the axes. The defaults are `c(0.05,0)` for continuous variables, and `c(0,0.6)` for discrete variables.

## See Also

`scale_continuous` for continuous position scales.

## Examples

```
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 values without scaling.

## Usage

```
scale_colour_identity(..., guide = "none")

scale_fill_identity(..., guide = "none")

scale_shape_identity(..., guide = "none")

scale_linetype_identity(..., guide = "none")

scale_alpha_identity(..., guide = "none")

scale_size_identity(..., guide = "none")
```

## Arguments

| | |
|---|---|
| ... | Other arguments passed on to discrete_scale or continuous_scale |
| guide | Guide to use for this scale - defaults to "none". |

## Examples

```
ggplot(luv_colours, aes(u, v)) +
  geom_point(aes(colour = col), size = 3) +
  scale_color_identity() +
  coord_equal()

df <- data.frame(
  x = 1:4,
  y = 1:4,
  colour = c("red", "green", "blue", "yellow")
)
ggplot(df, aes(x, y)) + geom_tile(aes(fill = colour))
ggplot(df, aes(x, y)) +
  geom_tile(aes(fill = colour)) +
  scale_fill_identity()

# To get a legend guide, specify guide = "legend"
ggplot(df, aes(x, y)) +
  geom_tile(aes(fill = colour)) +
  scale_fill_identity(guide = "legend")
# But you'll typically also need to supply breaks and labels:
ggplot(df, aes(x, y)) +
```

```
  geom_tile(aes(fill = colour)) +
  scale_fill_identity("trt", labels = letters[1:4], breaks = df$colour,
  guide = "legend")

# cyl scaled to appropriate size
ggplot(mtcars, aes(mpg, wt)) + geom_point(aes(size = cyl))

# cyl used as point size
ggplot(mtcars, aes(mpg, wt)) +
  geom_point(aes(size = cyl)) +
  scale_size_identity()
```

---

`scale_linetype` *Scale for line patterns.*

---

## Description

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

## Usage

```
scale_linetype(..., na.value = "blank")

scale_linetype_continuous(...)

scale_linetype_discrete(..., na.value = "blank")
```

## Arguments

| | |
|---|---|
| `...` | common discrete scale parameters: `name`, `breaks`, `labels`, `na.value`, `limits` and `guide`. See `discrete_scale` for more details |
| `na.value` | The linetype to use for `NA` values. |

## Examples

```
base <- ggplot(economics_long, aes(date, value01))
base + geom_line(aes(group = variable))
base + geom_line(aes(linetype = variable))

# See scale_manual for more flexibility
```

scale_manual *Create your own discrete scale.*

## Description

Create your own discrete scale.

## Usage

```
scale_colour_manual(..., values)

scale_fill_manual(..., values)

scale_size_manual(..., values)

scale_shape_manual(..., values)

scale_linetype_manual(..., values)

scale_alpha_manual(..., values)
```

## Arguments

| | |
|---|---|
| ... | common discrete scale parameters: name, breaks, labels, na.value, limits and guide. See discrete_scale for more details |
| values | a set of aesthetic values to map data values to. If this is a named vector, then the values will be matched based on the names. If unnamed, values will be matched in order (usually alphabetical) with the limits of the scale. Any data values that don't match will be given na.value. |

## Examples

```
p <- ggplot(mtcars, aes(mpg, wt)) +
  geom_point(aes(colour = factor(cyl)))

p + scale_colour_manual(values = c("red","blue", "green"))
p + scale_colour_manual(
  values = c("8" = "red","4" = "blue","6" = "green"))
# With rgb hex values
p + scale_colour_manual(values = c("#FF0000", "#0000FF", "#00FF00"))

# As with other scales you can use breaks to control the appearance
# of the legend
cols <- c("8" = "red","4" = "blue","6" = "darkgreen", "10" = "orange")
p + scale_colour_manual(values = cols)
p + scale_colour_manual(values = cols, breaks = c("4", "6", "8"))
p + scale_colour_manual(values = cols, breaks = c("8", "6", "4"))
```

```
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"))

# Notice that the values are matched with limits, and not breaks
p + scale_colour_manual(limits = c(6, 8, 4), breaks = c(8, 4, 6),
  values = c("grey50", "grey80", "black"))
```

---

`scale_shape` *Scale for shapes, aka glyphs.*

---

## Description

A continuous variable can not be mapped to shape.

## Usage

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

## Arguments

| | |
|---|---|
| `...` | common discrete scale parameters: `name`, `breaks`, `labels`, `na.value`, `limits` and `guide`. See `discrete_scale` for more details |
| `solid` | Are the shapes 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")
d + scale_shape(name = "Cut of\ndiamond")

# 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))

# Or for short:
d %+% dsmall
```

scale_size *Scale size (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 NULL, the default, the name of the scale is taken from the first mapping used for that aesthetic. |
| breaks | One of:<br>• NULL for no breaks<br>• waiver() for the default breaks computed by the transformation object<br>• A numeric vector of positions<br>• A function that takes the limits as input and returns breaks as output |
| labels | One of:<br>• NULL for no labels<br>• waiver() for the default labels computed by the transformation object<br>• A character vector giving labels (must be same length as breaks)<br>• 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".<br>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. `boxcox_trans`. You can create your own transformation with `trans_new`. |
| `guide` | Name of guide object, or object itself. |
| `...` | Other arguments passed on to `continuous_scale` to control name, limits, breaks, labels and so forth. |
| `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

```
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_size_animint` *Scale point sizes using circle area, but specifying the radius in pixels.*

---

### Description

Scale point sizes using circle area, but specifying the radius in pixels.

### Usage

```
scale_size_animint(pixel.range = c(2, 20), ...)
```

### Arguments

| | |
|---|---|
| `pixel.range` | min and max circle radius in pixels. |
| `...` | passed to continuous_scale. |

scale_x_discrete *Discrete position.*

## 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())

scale_y_discrete(..., expand = waiver())
```

## Arguments

| | |
|---|---|
| ... | common discrete scale parameters: name, breaks, labels, na.value, limits and guide. See discrete_scale for more details |
| expand | a numeric vector of length two giving multiplicative and additive expansion constants. These constants ensure that the data is placed some distance away from the axes. |

## Examples

```
ggplot(diamonds, aes(cut)) + geom_bar()


# The discrete position scale is added automatically whenever you
# have a discrete position.

(d <- ggplot(subset(diamonds, carat > 1), aes(cut, clarity)) +
      geom_jitter())

d + scale_x_discrete("Cut")
d + scale_x_discrete("Cut", labels = c("Fair" = "F","Good" = "G",
  "Very Good" = "VG","Perfect" = "P","Ideal" = "I"))

# Use limits to adjust the which levels (and in what order)
# are displayed
d + scale_x_discrete(limits = c("Fair","Ideal"))

# you can also use the short hand functions xlim and ylim
d + xlim("Fair","Ideal", "Good")
d + ylim("I1", "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 *Vector field of seal movements.*

---

### Description

This vector field was produced from the data described in Brillinger, D.R., Preisler, H.K., Ager, A.A. and Kie, J.G. "An exploratory data analysis (EDA) of the paths of moving animals". J. Statistical Planning and Inference 122 (2004), 43-63, using the methods of Brillinger, D.R., "Learning a potential function from a trajectory", Signal Processing Letters. December (2007).

### Usage

```
seals
```

### Format

A data frame with 1155 rows and 4 variables

### References

http://www.stat.berkeley.edu/~brill/Papers/jspifinal.pdf

---

## selectSSandCS *Separate .variable/.value selectors*

---

### Description

Separate .variable/.value selectors

### Usage

```
selectSSandCS(aesthetics_list)
```

### Arguments

| | |
|---|---|
| aesthetics_list | aesthetics mapping of the layer |

### Value

Modified `aes.list` list with separated showSelected.variable/value

---

`setPlotSizes` *Set plot width and height for all plots*

---

### Description

Set plot width and height for all plots

### Usage

```
setPlotSizes(meta, AllPlotsInfo)
```

### Arguments

| | |
|---|---|
| `meta` | meta object with all information |
| `AllPlotsInfo` | plot info list |

### Value

`NULL`. Sizes are stored in meta object

---

`split.x` *Split data.frame into recursive list of data.frame.*

---

### Description

Split data.frame into recursive list of data.frame.

### Usage

```
## S3 method for class 'x'
split(x, vars)
```

### Arguments

| | |
|---|---|
| `x` | data.frame. |
| `vars` | character vector of variable names to split on. |

### Value

recursive list of data.frame.

stat_ecdf *Empirical Cumulative Density Function*

## Description

Empirical Cumulative Density Function

## 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:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>A function will be called with a single argument, the plot data. The return value must be a data.frame., and will be used as the layer data. |
| geom | The geometric object to use display the data |
| position | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| ... | other arguments passed on to layer. These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat. |
| n | if NULL, do not interpolate. If not NULL, this is the number of points to interpolate with. |
| pad | If TRUE, pad the ecdf with additional points (-Inf, 0) and (Inf, 1) |
| na.rm | If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |

### Computed variables

**x** x in data

**y** cumulative density corresponding x

### Examples

```
df <- data.frame(x = rnorm(1000))
ggplot(df, aes(x)) + stat_ecdf(geom = "step")

df <- data.frame(x = c(rnorm(100, 0, 3), rnorm(100, 0, 10)),
                 g = gl(2, 100))

ggplot(df, aes(x, colour = g)) + stat_ecdf()
```

---

## stat_ellipse *Plot data ellipses.*

---

### Description

The method for calculating the ellipses has been modified from `car::ellipse` (Fox and Weisberg, 2011)

### Usage

```
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:<br>If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot`.<br>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.<br>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. |
| type | The type of ellipse. The default "t" assumes a multivariate t-distribution, and "norm" assumes a multivariate normal distribution. "euclid" draws a circle with the radius equal to level, representing the euclidean distance from the center. This ellipse probably won't appear circular unless coord_fixed() is applied. |
| level | The confidence level at which to draw an ellipse (default is 0.95), or, if type="euclid", the radius of the circle to be drawn. |
| segments | The number of segments to be used in drawing the ellipse. |
| na.rm | If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |

## References

John Fox and Sanford Weisberg (2011). An R Companion to Applied Regression, Second Edition. Thousand Oaks CA: Sage. URL: http://socserv.socsci.mcmaster.ca/jfox/Books/Companion

## 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` *Superimpose a function.*

---

## Description

Superimpose a function.

## Usage

```
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:<br>If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot`.<br>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.<br>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 |
| `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), removes missing values with a warning. If `TRUE` silently removes missing values. |
| `show.legend` | logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. |

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

## Aesthetics

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

- y

## Computed variables

**x** x's along a grid

**y** value of function evaluated at corresponding x

## Examples

```
set.seed(1492)
df <- data.frame(
  x = rnorm(100)
)
x <- df$x
base <- ggplot(df, aes(x)) + geom_density()
base + stat_function(fun = dnorm, colour = "red")
base + stat_function(fun = dnorm, colour = "red", args = list(mean = 3))

# Plot functions without data
# Examples adapted from Kohske Takahashi

# Specify range of x-axis
ggplot(data.frame(x = c(0, 2)), aes(x)) +
  stat_function(fun = exp, geom = "line")

# Plot a normal curve
ggplot(data.frame(x = c(-5, 5)), aes(x)) + stat_function(fun = dnorm)

# To specify a different mean or sd, use the args parameter to supply new values
ggplot(data.frame(x = c(-5, 5)), aes(x)) +
  stat_function(fun = dnorm, args = list(mean = 2, sd = .5))

# Two functions on the same plot
f <- ggplot(data.frame(x = c(0, 10)), aes(x))
f + stat_function(fun = sin, colour = "red") +
  stat_function(fun = cos, colour = "blue")

# Using a custom function
test <- function(x) {x ^ 2 + x + 20}
f + stat_function(fun = test)
```

stat_identity *Identity statistic.*

## Description

The identity statistic leaves the data unchanged.

## Usage

```
stat_identity(mapping = NULL, data = NULL, geom = "point",
  position = "identity", ..., show.legend = NA, inherit.aes = TRUE)
```

## Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by aes or aes_. If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping. |
| data | The data to be displayed in this layer. There are three options:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>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. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |

## Examples

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

stat_qq *Calculation for quantile-quantile plot.*

## Description

Calculation for quantile-quantile plot.

## Usage

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

## 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:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>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. |
| distribution | Distribution function to use, if x not specified |
| dparams | Additional parameters passed on to distribution function. |
| na.rm | If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |

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

## Aesthetics

stat_qqunderstands the following aesthetics (required aesthetics are in bold):

- **sample**
- x
- y

## Computed variables

**sample** sample quantiles

**theoretical** theoretical quantiles

## Examples

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

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

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

---

# 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:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>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), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |

### 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 x.*

---

### Description

stat_summary operates on unique x; stat_summary_bin operators on binned x. They are more flexible versions of stat_bin: instead of just counting, they can compute any aggregate.

### Usage

```
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(),
  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:<br>If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.<br>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.<br>A function will be called with a single argument, the plot data. The return value must be a data.frame., and will be used as the layer data. |
| geom | Use to override the default connection between geom_histogram/geom_freqpoly and stat_bin. |
| position | Position adjustment, either as a string, or the result of a call to a position adjustment function. |
| ... | other arguments passed on to layer. These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat. |
| fun.data | A function that is given the complete data and should return a data frame with variables ymin, y, and ymax. |
| fun.ymin, fun.y, fun.ymax | Alternatively, supply three individual functions that are each passed a vector of x's and should return a single number. |
| fun.args | Optional additional arguments passed on to the functions. |
| na.rm | If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders. |

## Aesthetics

stat_summaryunderstands the following aesthetics (required aesthetics are in bold):

- **x**
- **y**

## 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 suppled, 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()

# 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
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("mean_sdl", fun.args = list(mult = 1), mapping = aes(group = cyl))
d + stat_sum_df("median_hilow", mapping = aes(group = cyl))

# An example with highly skewed distributions:
if (require("ggplot2movies")) {
set.seed(596)
mov <- movies[sample(nrow(movies), 1000), ]
 m2 <- ggplot(mov, aes(x = factor(round(rating)), y = votes)) + geom_point()
 m2 <- m2 + stat_summary(fun.data = "mean_cl_boot", geom = "crossbar",
                         colour = "red", width = 0.3) + xlab("rating")
m2
# Notice how the overplotting skews off visual perception of the mean
# supplementing the raw data with summary statistics is _very_ important

# Next, we'll look at votes on a log scale.

# Transforming the scale means the data are transformed
# first, after which statistics are computed:
m2 + scale_y_log10()
# Transforming the coordinate system occurs after the
# statistic has been computed. This means we're calculating the summary on the raw data
# and stretching the geoms onto the log scale.  Compare the widths of the
# standard errors.
m2 + coord_trans(y="log10")
}
```

---

`stat_unique` *Remove duplicates.*

---

## Description

Remove duplicates.

## Usage

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

## Arguments

| | |
|---|---|
| `mapping` | Set of aesthetic mappings created by `aes` or `aes_`. If specified and `inherit.aes` = `TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping. |
| `data` | The data to be displayed in this layer. There are three options:<br>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.<br>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), removes missing values with a warning. If `TRUE` silently removes missing values. |
| `show.legend` | logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. |
| `inherit.aes` | If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders`. |

## Aesthetics

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

- 

## Examples

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

---

# `switch_axes` *Flip axes in case of coord_flip Switches column names. Eg. xmin to ymin, yntercept to xintercept etc.*

---

## Description

Flip axes in case of coord_flip Switches column names. Eg. xmin to ymin, yntercept to xintercept etc.

## Usage

```
switch_axes(col.names)
```

## Arguments

| | |
|---|---|
| `col.names` | Column names which need to be switched |

### Value

Column names with x and y axes switched

---

`theme` *Set theme elements*

---

### Description

Use this function to modify theme settings.

### Usage

```
theme(..., complete = FALSE, validate = TRUE)
```

### Arguments

| | |
|---|---|
| `...` | a list of element name, element pairings that modify the existing theme. |
| `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. |
| `validate` | TRUE to run validate_element, FALSE to bypass checks. |

### Details

Theme elements can 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`.

For more examples of modifying properties using inheritance, `%+replace%`.

To see a graphical representation of the inheritance tree, see the last example below.

### Theme elements

The individual theme elements are:

| | |
|---|---|
| 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.y | y axis label (`element_text`; inherits from `axis.title`) |
| 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.y | y axis tick labels (`element_text`; inherits from `axis.text`) |
| axis.ticks | tick marks along axes (`element_line`; inherits from `line`) |

| | |
|---|---|
| axis.ticks.x | x axis tick marks (element_line; inherits from axis.ticks) |
| axis.ticks.y | y axis tick marks (element_line; inherits from axis.ticks) |
| axis.ticks.length | length of tick marks (unit) |
| axis.line | lines along axes (element_line; inherits from line) |
| axis.line.x | line along x axis (element_line; inherits from axis.line) |
| axis.line.y | line along y axis (element_line; inherits from axis.line) |
| legend.background | background of legend (element_rect; inherits from rect) |
| legend.margin | extra space added around legend (unit) |
| legend.key | background underneath legend keys (element_rect; inherits from rect) |
| legend.key.size | size of legend keys (unit; inherits from legend.key.size) |
| 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) |
| 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", " |
| 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 shou |
| panel.margin | margin around facet panels (unit) |
| panel.margin.x | horizontal margin around facet panels (unit; inherits from panel.margin) |
| panel.margin.y | vertical margin around facet panels (unit; inherits from panel.margin) |
| 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 |
| 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 de |
| 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.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) |

## See Also

%+replace%

rel

element_blank

element_line

element_rect

element_text

## Examples

```
p <- ggplot(mtcars, aes(mpg, wt)) +
  geom_point()
p
p + theme(panel.background = element_rect(colour = "pink"))
p + theme_bw()

# Scatter plot of gas mileage by vehicle weight
p <- ggplot(mtcars, aes(wt, mpg)) +
  geom_point()
# Calculate slope and intercept of line of best fit
coef(lm(mpg ~ wt, data = mtcars))
p + geom_abline(intercept = 37, slope = -5)
# Calculate correlation coefficient
with(mtcars, cor(wt, mpg, use = "everything", method = "pearson"))
#annotate the plot
p + geom_abline(intercept = 37, slope = -5) +
geom_text(data = data.frame(), aes(4.5, 30, label = "Pearson-R = -.87"))

# Change the axis labels
# Original plot
p
p + labs(x = "Vehicle Weight", y = "Miles per Gallon")
# Or
p + labs(x = "Vehicle Weight", y = "Miles per Gallon")

# Change title appearance
p <- p + labs(title = "Vehicle Weight-Gas Mileage Relationship")
# Set title to twice the base font size
p + theme(plot.title = element_text(size = rel(2)))
p + theme(plot.title = element_text(size = rel(2), colour = "blue"))

# Add a subtitle and adjust bottom margin
p + labs(title = "Vehicle Weight-Gas Mileage Relationship",
         subtitle = "You need to wrap long subtitleson manually") +
    theme(plot.subtitle = element_text(margin = margin(b = 20)))

# Changing plot look with themes
DF <- data.frame(x = rnorm(400))
m <- ggplot(DF, aes(x = x)) +
```

```
  geom_histogram()
# Default is theme_grey()
m
# Compare with
m + theme_bw()

# Manipulate Axis Attributes
m + theme(axis.line = element_line(size = 3, colour = "red", linetype = "dotted"))
m + theme(axis.text = element_text(colour = "blue"))
m + theme(axis.text.y = element_blank())
m + theme(axis.ticks = element_line(size = 2))
m + theme(axis.title.y = element_text(size = rel(1.5), angle = 90))
m + theme(axis.title.x = element_blank())
m + theme(axis.ticks.length = unit(.85, "cm"))

# Legend Attributes
z <- ggplot(mtcars, aes(wt, mpg)) +
  geom_point(aes(colour = factor(cyl)))
z
z + theme(legend.position = "none")
z + theme(legend.position = "bottom")
# Or use relative coordinates between 0 and 1
z + theme(legend.position = c(.5, .5))
# Add a border to the whole legend
z + theme(legend.background = element_rect(colour = "black"))
# Legend margin controls extra space around outside of legend:
z + theme(legend.background = element_rect(),
          legend.margin = unit(1, "cm"))
z + theme(legend.background = element_rect(),
          legend.margin = unit(0, "cm"))
# Or to just the keys
z + theme(legend.key = element_rect(colour = "black"))
z + theme(legend.key = element_rect(fill = "yellow"))
z + theme(legend.key.size = unit(2.5, "cm"))
z + theme(legend.text = element_text(size = 20, colour = "red", angle = 45))
z + theme(legend.title = element_text(face = "italic"))

# To change the title of the legend use the name argument
# in one of the scale options
z + scale_colour_brewer(name = "My Legend")
z + scale_colour_grey(name = "Number of \nCylinders")

# Panel and Plot Attributes
z + theme(panel.background = element_rect(fill = "black"))
z + theme(panel.border = element_rect(linetype = "dashed", colour = "black"))
z + theme(panel.grid.major = element_line(colour = "blue"))
z + theme(panel.grid.minor = element_line(colour = "red", linetype = "dotted"))
z + theme(panel.grid.major = element_line(size = 2))
z + theme(panel.grid.major.y = element_blank(),
          panel.grid.minor.y = element_blank())
z + theme(plot.background = element_rect())
z + theme(plot.background = element_rect(fill = "green"))
```

```
# Faceting Attributes
set.seed(4940)
dsmall <- diamonds[sample(nrow(diamonds), 1000), ]
k <- ggplot(dsmall, aes(carat, ..density..)) +
  geom_histogram(binwidth = 0.2) +
  facet_grid(. ~ cut)
k + theme(strip.background = element_rect(colour = "purple", fill = "pink",
                                          size = 3, linetype = "dashed"))
k + theme(strip.text.x = element_text(colour = "red", angle = 45, size = 10,
                                      hjust = 0.5, vjust = 0.5))
k + theme(panel.margin = unit(5, "lines"))
k + theme(panel.margin.y = unit(0, "lines"))

# Put gridlines on top
meanprice <- tapply(diamonds$price, diamonds$cut, mean)
cut <- factor(levels(diamonds$cut), levels = levels(diamonds$cut))
df <- data.frame(meanprice, cut)
g <- ggplot(df, aes(cut, meanprice)) + geom_bar(stat = "identity")
g + geom_bar(stat = "identity") +
    theme(panel.background = element_blank(),
          panel.grid.major.x = element_blank(),
          panel.grid.minor.x = element_blank(),
          panel.grid.minor.y = element_blank(),
          panel.ontop = TRUE)

# Modify a theme and save it
mytheme <- theme_grey() + theme(plot.title = element_text(colour = "red"))
p + mytheme
```

---

`theme_animint` *theme for passing animint specific params*

---

## Description

Theme without checks. This allows us to write `theme_animint(width=500)`, instead of `theme(animint.width=500)` which gives an error in ggplot2 because users should be informed if they mis-type standard theme element names. https://github.com/hadley/ggplot2/issues/938

## Usage

```
theme_animint(...)
```

## Arguments

| | |
|---|---|
| `...` | theme options such as `width`. Use `update_axes=c("x","y")` to update the axes of plots. Works for single selection variables. |

### Value

ggplot theme list with names such as `animint.width`.

### Author(s)

Toby Dylan Hocking

### Examples

```
mtcars$cyl <- as.factor(mtcars$cyl)
p <- ggplot() +
  geom_point(aes(x=wt, y=mpg, colour=cyl),
             data=mtcars) +
  ## set width and height values and update both axes
  theme_animint(width=600, height=600, update_axes=c("x", "y"))
viz <- list(plot=p, selector.types=list(cyl="single"))
animint2dir(viz)
```

---

## `theme_update` *Get, set and update themes.*

---

### Description

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 in the current 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` will completely replace the element, so any unspecified values will overwrite the current value in the theme with `NULL`s.

### Usage

```
theme_update(...)

theme_replace(...)

theme_get()

theme_set(new)
```

### Arguments

| | |
|---|---|
| `...` | named list of theme settings |
| `new` | new theme (a list of theme elements) |

### See Also

`%+replace%`

## Examples

```
p <- ggplot(mtcars, aes(mpg, wt)) +
  geom_point()
p
old <- theme_set(theme_bw())
p
theme_set(old)
p

#theme_replace NULLs out the fill attribute of panel.background,
#resulting in a white background:
theme_get()$panel.background
old <- theme_replace(panel.background = element_rect(colour = "pink"))
theme_get()$panel.background
p
theme_set(old)

#theme_update only changes the colour attribute, leaving the others intact:
old <- theme_update(panel.background = element_rect(colour = "pink"))
theme_get()$panel.background
p
theme_set(old)

theme_get()


ggplot(mtcars, aes(mpg, wt)) +
  geom_point(aes(color = mpg)) +
  theme(legend.position = c(0.95, 0.95),
        legend.justification = c(1, 1))
last_plot() +
 theme(legend.background = element_rect(fill = "white", colour = "white", size = 3))
```

---

# toRGB *Convert R colors to RGB hexadecimal color values*

---

## Description

Convert R colors to RGB hexadecimal color values

## Usage

```
toRGB(x)
```

## Arguments

| | |
|---|---|
| x | character |

### Value

hexadecimal color value or "transparent" if is.na

---

`transform_shape` *Function to transform R shapes into d3 shapes...*

---

### Description

Function to transform R shapes into d3 shapes...

### Usage

```
transform_shape(dframe)
```

### Arguments

| | |
|---|---|
| `dframe` | Data frame with columns shape, fill, colour. |

### Value

Data frame transformed so that shape corresponds to d3 shape. Also includes Rshape column for debugging.

---

`translate_qplot_ggplot`
*Translating between qplot and ggplot*

---

### Description

Within ggplot2, there are two basic methods to create plots, with qplot() and ggplot(). qplot() is designed primarily for interactive use: it makes a number of assumptions that speed most cases, but when designing multilayered plots with different data sources it can get in the way. This section describes what those defaults are, and how they map to the fuller ggplot() syntax.

### Examples

```
# By default, qplot() assumes that you want a scatterplot,
# i.e., you want to use geom_point()
# qplot(x, y, data = data)
# ggplot(data, aes(x, y)) + geom_point()

# Using Aesthetics

# If you map additional aesthetics, these will be added to the defaults. With
# qplot() there is no way to use different aesthetic mappings (or data) in
```

```
# different layers
# qplot(x, y, data = data, shape = shape, colour = colour)
# ggplot(data, aes(x, y, shape = shape, colour = colour)) + geom_point()
#
# Aesthetic parameters in qplot() always try to map the aesthetic to a
# variable. If the argument is not a variable but a value, effectively a new column
# is added to the original dataset with that value. To set an aesthetic to a
# value and override the default appearance, you surround the value with I() in
# qplot(), or pass it as a parameter to the layer.
# qplot(x, y, data = data, colour = I("red"))
# ggplot(data, aes(x, y)) + geom_point(colour = "red")

# Changing the geom parameter changes the geom added to the plot
# qplot(x, y, data = data, geom = "line")
# ggplot(data, aes(x, y)) + geom_line()

# Not all geoms require both x and y, e.g., geom_bar() and geom_histogram().
# For these two geoms, if the y aesthetic is not supplied, both qplot and
# ggplot commands default to "count" on the y-axis
# ggplot(data, aes(x)) + geom_bar()
# qplot(x, data = data, geom = "bar")

# If a vector of multiple geom names is supplied to the geom argument, each
# geom will be added in turn
# qplot(x, y, data = data, geom = c("point", "smooth"))
# ggplot(data, aes(x, y)) + geom_point() + geom_smooth()

# Unlike the rest of ggplot2, stats and geoms are independent
# qplot(x, y, data = data, stat = "bin")
# ggplot(data, aes(x, y)) + geom_point(stat = "bin")
#
# Any layer parameters will be passed on to all layers. Most layers will ignore
# parameters that they don't need
# qplot(x, y, data = data, geom = c("point", "smooth"), method = "lm")
# ggplot(data, aes(x, y)) + geom_point(method = "lm") + geom_smooth(method = "lm")

# Scales and axes

# You can control basic properties of the x and y scales with the xlim, ylim,
# xlab and ylab arguments
# qplot(x, y, data = data, xlim = c(1, 5), xlab = "my label")
# ggplot(data, aes(x, y)) + geom_point() +
# scale_x_continuous("my label", limits = c(1, 5))

# qplot(x, y, data = data, xlim = c(1, 5), ylim = c(10, 20))
# ggplot(data, aes(x, y)) + geom_point() +
# scale_x_continuous(limits = c(1, 5)) + scale_y_continuous(limits = c(10, 20))

# Like plot(), qplot() has a convenient way of log transforming the axes.
# qplot(x, y, data = data, log = "xy")
# ggplot(data, aes(x, y)) + geom_point() + scale_x_log10() + scale_y_log10()
# There are many other possible transformations, but not all are
# accessible from within qplot(), see ?scale_continuous for more
```

```
# Plot options

# qplot() recognises the same options as plot does, and converts them to their
# ggplot2 equivalents. See ?theme for more on ggplot options
# qplot(x, y, data = data, main="title", asp = 1)
# ggplot(data, aes(x, y)) + geom_point() + labs(title = "title") + theme(aspect.ratio = 1)
```

---

`translate_qplot_lattice`

*Translating between qplot and lattice*

---

## Description

The major difference between lattice and ggplot2 is that lattice uses a formula based interface. ggplot2 does not because the formula does not generalise well to more complicated situations.

## Examples

```
library(lattice)

if (require("ggplot2movies")) {
xyplot(rating ~ year, data=movies)
qplot(year, rating, data=movies)

xyplot(rating ~ year | Comedy + Action, data = movies)
qplot(year, rating, data = movies, facets = ~ Comedy + Action)
# Or maybe
qplot(year, rating, data = movies, facets = Comedy ~ Action)

# While lattice has many different functions to produce different types of
# graphics (which are all basically equivalent to setting the panel argument),
# ggplot2 has qplot().

stripplot(~ rating, data = movies, jitter.data = TRUE)
qplot(rating, 1, data = movies, geom = "jitter")

histogram(~ rating, data = movies)
qplot(rating, data = movies, geom = "histogram")

bwplot(Comedy ~ rating ,data = movies)
qplot(factor(Comedy), rating, data = movies, geom = "boxplot")

xyplot(wt ~ mpg, mtcars, type = c("p","smooth"))
qplot(mpg, wt, data = mtcars, geom = c("point","smooth"))
}

# The capabilities for scale manipulations are similar in both ggplot2 and
# lattice, although the syntax is a little different.
```

```
xyplot(wt ~ mpg | cyl, mtcars, scales = list(y = list(relation = "free")))
qplot(mpg, wt, data = mtcars) + facet_wrap(~ cyl, scales = "free")

xyplot(wt ~ mpg | cyl, mtcars, scales = list(log = 10))
qplot(mpg, wt, data = mtcars, log = "xy")

xyplot(wt ~ mpg | cyl, mtcars, scales = list(log = 2))
qplot(mpg, wt, data = mtcars) +
  scale_x_continuous(trans = scales::log2_trans()) +
  scale_y_continuous(trans = scales::log2_trans())

xyplot(wt ~ mpg, mtcars, group = cyl, auto.key = TRUE)
# Map directly to an aesthetic like colour, size, or shape.
qplot(mpg, wt, data = mtcars, colour = cyl)

xyplot(wt ~ mpg, mtcars, xlim = c(20,30))
# Works like lattice, except you can't specify a different limit
# for each panel/facet
qplot(mpg, wt, data = mtcars, xlim = c(20,30))

# Both lattice and ggplot2 have similar options for controlling labels on the plot.

xyplot(wt ~ mpg, mtcars, xlab = "Miles per gallon", ylab = "Weight",
  main = "Weight-efficiency tradeoff")
qplot(mpg, wt, data = mtcars, xlab = "Miles per gallon", ylab = "Weight",
  main = "Weight-efficiency tradeoff")

xyplot(wt ~ mpg, mtcars, aspect = 1)
qplot(mpg, wt, data = mtcars, asp = 1)

# par.settings() is equivalent to + theme() and trellis.options.set()
# and trellis.par.get() to theme_set() and theme_get().
# More complicated lattice formulas are equivalent to rearranging the data
# before using ggplot2.
```

---

`txhousing` *Housing sales in TX.*

---

## Description

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

## Usage

```
txhousing
```

### Format

A data frame with 8602 observations and 9 variables:

- cityName of MLS area
- year,month,dateDate
- salesNumber of sales
- volumeTotal value of sales
- medianMedian sale price
- listingsTotal active listings
- inventory"Months inventory": amount of time it would take to sell all current listings at current pace of sales.

---

## update_geom_defaults

*Modify geom/stat aesthetic defaults for future plots*

---

### Description

Modify geom/stat aesthetic defaults for future plots

### Usage

```
update_geom_defaults(geom, new)

update_stat_defaults(stat, new)
```

### Arguments

| | |
|---|---|
| new | Named list of aesthetics. |
| stat, geom | Name of geom/stat to modify (like "point" or "bin"), or a Geom/Stat object (like GeomPoint or StatBin). |

### Examples

```
update_geom_defaults("point", list(colour = "darkblue"))
ggplot(mtcars, aes(mpg, wt)) + geom_point()
update_geom_defaults("point", list(colour = "black"))
```

## update_labels *Update axis/legend labels*

### Description

Update axis/legend labels

### Usage

```
update_labels(p, labels)
```

### Arguments

| | |
|---|---|
| p | plot to modify |
| labels | named list of new labels |

### Examples

```
p <- ggplot(mtcars, aes(mpg, wt)) + geom_point()
update_labels(p, list(x = "New x"))
update_labels(p, list(x = expression(x / y ^ 2)))
update_labels(p, list(x = "New x", y = "New Y"))
update_labels(p, list(colour = "Fail silently"))
```

## UStornadoes *Tornadoes in the United States from 1950 to 2012*

### Description

Each row documents 1 tornado.

### Usage

```
data(UStornadoes)
```

### Format

A data frame with 41620 observations on the following 32 variables.

`fips` a numeric vector

`ID` a numeric vector

`year` a numeric vector

`month` a numeric vector

`day` a numeric vector

`date` factor

`time` a numeric vector

`tz` a numeric vector

`state` factor

`state.tnum` a numeric vector

`f` a numeric vector

`injuries` a numeric vector

`fatalities` a numeric vector

`propertyLoss` a numeric vector

`cropLoss` a numeric vector

`startLat` a numeric vector

`startLong` a numeric vector

`endLat` a numeric vector

`endLong` a numeric vector

`trackLength` a numeric vector

`trackWidth` a numeric vector

`numStatesAffected` a numeric vector

`stateNumber` a numeric vector

`segmentNumber` a numeric vector

`FipsCounty1` a numeric vector

`FipsCounty2` a numeric vector

`FipsCounty3` a numeric vector

`FipsCounty4` a numeric vector

`TotalPop2012` a numeric vector

`LandArea` a numeric vector

`TornadoesSqMile` a numeric vector

`weight` a numeric vector

## Source

NOAA SVRGIS data (Severe Report Database + Geographic Information System) http://www.spc.noaa.gov/gis/svrgis/

## varied.chunk *Extract subset for each data.frame in a list of data.frame*

### Description

Extract subset for each data.frame in a list of data.frame

### Usage

```
varied.chunk(df.or.list, cols)
```

### Arguments

| | |
|---|---|
| df.or.list | a data.frame or a list of data.frame. |
| cols | cols that each data.frame would keep. |

### Value

list of data.frame.

## WorldBank *Demographics by country from 1960 to 2012*

### Description

Each row is one year of demographics for one country.

### Usage

```
data(WorldBank)
```

### Format

A data frame with 11342 observations on the following 15 variables.

iso2c a character vector

country a character vector

year a numeric vector

fertility.rate a numeric vector

life.expectancy a numeric vector

population a numeric vector

GDP.per.capita.Current.USD a numeric vector

15.to.25.yr.female.literacy a numeric vector

`iso3c` factor

`region` factor

`capital` factor

`longitude` factor

`latitude` factor

`income` factor

`lending` factor

## Source

Copied from the googleVis package.

---

`worldPop` *World population by subcontinent*

---

## Description

World population data are used as a simple example on the polychart.js website, and so these data can be used to recreate that example using animint.

## Usage

```
data(worldPop)
```

## Format

A data frame with 294 observations on the following 4 variables.

`subcontinent` factor: the subcontinent name

`year` integer: year of measurement

`population` integer: number of people in that subcontinent during that year

`type` factor with levels `actual` `estimate`

## Source

https://github.com/Polychart/polychart2/blob/master/example/population.coffee