

# Data visualization with ggplot2 Cheat Sheet

Gustavo Santos Paiva Laender Moura

2025-03-01



```
library(ggplot2)
```

## Overview

- Run `?ggplot2` for author information. ggplot2 is a system for ‘declaratively’ creating graphics, based on “The Grammar of Graphics”. You provide the data, tell ‘ggplot2’ how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.

- For package documentation, run `help(package = "ggplot2")`
- Need help getting started? Try the [R Graphics Cookbook](#)
- [Tidyverse::ggplot2](#)
- [Github](#)
- [Cheat sheet](#)
- [Cheat sheet pdf](#)
- Run `??ggplot2` for broader search
- For an html helper page on your browser, run,,

```
help.start()
```

If the browser launched by `'/usr/bin/open'` is already running, it is  
 \*not\* restarted, and you must switch to its window.  
 Otherwise, be patient ...

```
browseVignettes("ggplot2")
```

Index

```
?ggplot2  
??ggplot2
```

## Datasets

The datasets used in the examples are: `fruit` (80 fruit names), `words` (980 words), and `sentences` (720 sentences). Run `?fruit` for documentation.

## Basics

**ggplot2** is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and **geoms**—visual marks that represent data points.

```
library(ggplot2)
```

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

Complete the template below to build a graph.

```
ggplot(data = <Data>) +  
  <Geom_Function>(mapping = aes(<Mappings>),  
    stat = <Stat>,  
    position = <Position>) +  
  <Coordinate_Function> +  
  <Facet_Function> +  
  <Scale_Function> +  
  <Theme_Function>
```

Data, a Geom Function, and Aes Mappings are required. Stat, Position, and the Coordinate, Facet, Scale, and Theme functions are not required and will supply sensible defaults.

- `ggplot(data = mpg, aes(x = cty, y = hwy))`: Begins a plot that you finish by adding layers to. Add one geom function per layer.
- `last_plot()`: Returns the last plot.
- `ggsave("plot.png", width = 5, height = 5)`: Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

## Aes

Common aesthetic values.

- **color** and **fill**: String ("red", "#RRGGBB").
- **linetype**: Integer or string (0 = "blank", 1 = "solid", 2 = "dashed", 3 = "dotted", 4 = "dotdash", 5 = "longdash", 6 = "twodash").
- **size**: Integer (in mm for size of points and text).

- **linewidth**: Integer (in mm for widths of lines).
- **shape**: Integer/shape name or a single character ("a").
  - shape integer/name pairs: 0 = "square open", 1 = "circle open", 2 = "triangle open", 3 = "plus", 4 = "cross", 5 = "diamond open", 6 = "triangle down open", 7 = "square cross", 8 = "asterisk", 9 = "diamond plus", 10 = "circle plus", 11 = "star", 12 = "square plus", 13 = "circle cross", 14 = "square triangle", 15 = "square", 16 = "circle", 17 = "triangle", 18 = "diamond", 19 = "circle small", 20 = "bullet", 21 = "circle filled", 22 = "square filled", 23 = "diamond filled", 24 = "triangle filled", 25 = "triangle down filled"

## Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

## Graphical Primitives

```
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))
```

- **a + geom\_blank()** and **a + expand\_limits()**: Ensure limits include values across all plots.
- **b + geom\_curve(aes(yend = lat + 1, xend = long + 1), curvature = 1)**: Draw a curved line from (x, y) to (xend, yend). **aes()** arguments: x, xend, y, yend, alpha, angle, color, curvature, linetype, size.
- **a + geom\_path(lineend = "butt", linejoin = "round", linemitre = 1)**: Connect observations in the order they appear. **aes()** arguments: x, y, alpha, color, group, linetype, size.
- **a + geom\_polygon(aes(alpha = 50))**: Connect points into polygons. **aes()** arguments: x, y, alpha, color, fill, group, subgroup, linetype, size.
- **b + geom\_rect(aes(xmin = long, ymin = lat, xmax = long + 1, ymax = lat + 1))**: Draw a rectangle by connecting four corners (xmin, xmax, ymin, ymax). **aes()** arguments: xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size.

- `a + geom_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900))`: For each `x`, plot an interval from `ymin` to `ymax`. `aes()` arguments: `x`, `ymax`, `ymin`, `alpha`, `color`, `fill`, `group`, `linetype`, `size`.

## Line Segments

Common aesthetics: `x`, `y`, `alpha`, `color`, `linetype`, `size`, `linewidth`.

- `b + geom_abline(aes(intercept = 0, slope = 1))`: Draw a diagonal reference line with a given slope and intercept.
- `b + geom_hline(aes(yintercept = lat))`: Draw a horizontal reference line with a given `yintercept`.
- `b + geom_vline(aes(xintercept = long))`: Draw a vertical reference line with a given `xintercept`.
- `b + geom_segment(aes(yend = lat + 1, xend = long + 1))`: Draw a straight line from `(x, y)` to `(xend, yend)`.
- `b + geom_spoke(aes(angle = 1:1155, radius = 1))`: Draw line segments using polar coordinates (`angle` and `radius`).

## One Variable - Continuous

```
c <- ggplot(mpg, aes(hwy))
c2 <- ggplot(mpg)
```

- `c + geom_area(stat = "bin")`: Draw an area plot. `aes()` arguments: `x`, `y`, `alpha`, `color`, `fill`, `linetype`, `linewidth`.
- `c + geom_density(kernel = "gaussian")`: Compute and draw kernel density estimates. `aes()` arguments: `x`, `y`, `alpha`, `color`, `fill`, `group`, `linetype`, `linewidth`, `weight`.
- `c + geom_dotplot()`: Draw a dot plot. `aes()` arguments: `x`, `y`, `alpha`, `color`, `fill`.
- `c + geom_freqpoly()`: Draw a frequency polygon. `aes()` arguments: `x`, `y`, `alpha`, `color`, `group`, `linetype`, `linewidth`.
- `c + geom_histogram(binwidth = 5)`: Draw a histogram. `aes()` arguments: `x`, `y`, `alpha`, `color`, `fill`, `linetype`, `linewidth`, `weight`.
- `c2 + geom_qq(aes(sample = hwy))`: Draw a quantile-quantile plot. `aes()` arguments: `x`, `y`, `alpha`, `color`, `fill`, `linetype`, `size`, `weight`.

## One Variable - Discrete

```
d <- ggplot(mpg, aes(fl))
```

- `d + geom_bar()`: Draw a bar chart. `aes()` arguments: `x`, `alpha`, `color`, `fill`, `linetype`, `linewidth`, `weight`.

## Two Variables - Both Continuous

```
e <- ggplot(mpg, aes(cty, hwy))
```

- `e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1)`: Add text with a rectangle background. `aes()` arguments: `x`, `y`, `label`, `alpha`, `angle`, `color`, `family`, `fontface`, `hjust`, `lineheight`, `size`, `vjust`.
- `e + geom_point()`: Draw a scatter plot. `aes()` arguments: `x`, `y`, `alpha`, `color`, `fill`, `shape`, `size`, `stroke`.
- `e + geom_quantile()`: Fit and draw quantile regression for the plot data. `aes()` arguments: `x`, `y`, `alpha`, `color`, `group`, `linetype`, `linewidth`, `weight`.
- `e + geom_rug(sides = "bl")`: Draw a rug plot. `aes()` arguments: `x`, `y`, `alpha`, `color`, `linetype`, `linewidth`.
- `e + geom_smooth(method = lm)`: Plot smoothed conditional means. `aes()` arguments: `x`, `y`, `alpha`, `color`, `fill`, `group`, `linetype`, `linewidth`, `weight`.
- `e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1)`: Add text to a plot. `aes()` arguments: `x`, `y`, `label`, `alpha`, `angle`, `color`, `family`, `fontface`, `hjust`, `lineheight`, `size`, `vjust`.

## Two Variables - One Discrete, One Continuous

```
f <- ggplot(mpg, aes(class, hwy))
```

- `f + geom_col()`: Draw a bar plot. `aes()` arguments: `x`, `y`, `alpha`, `color`, `fill`, `group`, `linetype`, `linewidth`.
- `f + geom_boxplot()`: Draw a box plot. `aes()` arguments: `x`, `y`, `lower`, `middle`, `upper`, `ymax`, `ymin`, `alpha`, `color`, `fill`, `group`, `linetype`, `shape`, `linewidth`, `weight`.

- `f + geom_dotplot(binaxis = "y", stackdir = "center")`: Draw a dot plot. `aes()` arguments: `x`, `y`, `alpha`, `color`, `fill`, `group`.
- `f + geom_violin(scale = "area")`: Draw a violin plot. `aes()` arguments: `x`, `y`, `alpha`, `color`, `fill`, `group`, `linetype`, `linewidth`, `weight`.

## Two Variables - Both Discrete

```
g <- ggplot(diamonds, aes(cut, color))
```

- `g + geom_count()`: Plot a count of points in an area to address over plotting. `aes()` arguments: `x`, `y`, `alpha`, `color`, `fill`, `shape`, `size`, `stroke`.
- `e + geom_jitter(height = 2, width = 2)`: Jitter points in a plot. `aes()` arguments: `x`, `y`, `alpha`, `color`, `fill`, `shape`, `size`.

## Two Variables - Continuous Bivariate Distribution

```
h <- ggplot(diamonds, aes(carat, price))
```

- `h + geom_bin2d(binwidth = c(0.25, 500))`: Draw a heatmap of 2D rectangular bin counts. `aes()` arguments: `x`, `y`, `alpha`, `color`, `fill`, `linetype`, `size`, `weight`.
- `h + geom_density_2d()`: Plot contours from 2D kernel density estimation. `aes()` arguments: `x`, `y`, `alpha`, `color`, `group`, `linetype`, `linewidth`.
- `h + geom_hex()`: Draw a heatmap of 2D hexagonal bin counts. `aes()` arguments: `x`, `y`, `alpha`, `color`, `fill`, `linewidth`.

## Two Variables - Continuous Function

```
i <- ggplot(economics, aes(date, unemploy))
```

- `i + geom_area()`: Draw an area plot. `aes()` arguments: `x`, `y`, `alpha`, `color`, `fill`, `linetype`, `linewidth`.
- `i + geom_line()`: Connect data points, ordered by the `x` axis variable. `aes()` arguments: `x`, `y`, `alpha`, `color`, `group`, `linetype`, `linewidth`.
- `i + geom_step(direction = "hv")`: Draw a stairstep plot. `aes()` arguments: `x`, `y`, `alpha`, `color`, `group`, `linetype`, `linewidth`.

## Two Variables - Visualizing Error

```
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))
```

- `j + geom_crossbar(fatten = 2)`: Draw a crossbar. `aes()` arguments: `x`, `y`, `ymax`, `ymin`, `alpha`, `color`, `fill`, `group`, `linetype`, `linewidth`.
- `j + geom_errorbar()`: Draw an errorbar. Also `geom_errorbarh()`. `aes()` arguments: `x`, `ymax`, `ymin`, `alpha`, `color`, `group`, `linetype`, `linewidth`, `width`.
- `j + geom_linerange()`: Draw a line range. `aes()` arguments: `x`, `ymin`, `ymax`, `alpha`, `color`, `group`, `linetype`, `linewidth`.
- `j + geom_pointrange()`: Draw a point range. `aes()` arguments: `x`, `y`, `ymin`, `ymax`, `alpha`, `color`, `fill`, `group`, `linetype`, `shape`, `linewidth`.

## Two Variables - Maps

Draw the appropriate geometric object depending on the simple features present in the data. `aes()` arguments: `map_id`, `alpha`, `color`, `fill`, `linetype`, `linewidth`.

```
nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"))
```

Reading layer `nc' from data source

  `/Users/gustavosplmoura/Library/R/arm64/4.4/library/sf/shape/nc.shp'

  using driver `ESRI Shapefile'

Simple feature collection with 100 features and 14 fields

Geometry type: MULTIPOLYGON

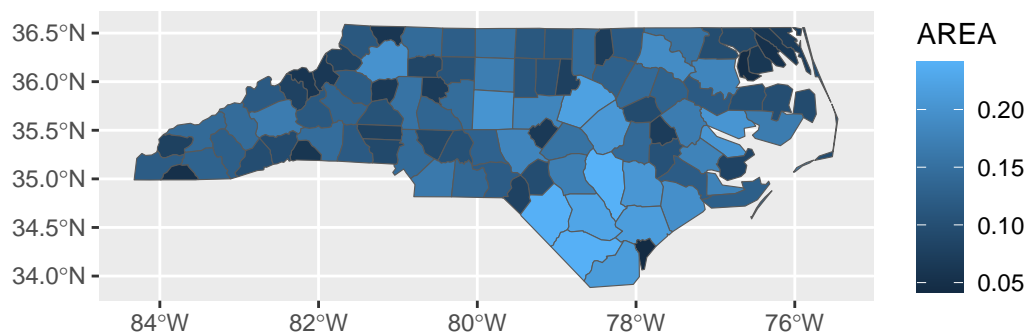
Dimension: XY

Bounding box: xmin: -84.32385 ymin: 33.88199 xmax: -75.45698 ymax: 36.58965

Geodetic CRS: NAD27

```
ggplot(nc) +
  geom_sf(aes(fill = AREA))
```





### Three Variables

```
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
l <- ggplot(seals, aes(long, lat))
```

- `l + geom_contour(aes(z = z))`: Draw 2D contour plot. `aes()` arguments: `x`, `y`, `z`, `alpha`, `color`, `group`, `linetype`, `linewidth`, `weight`.
- `l + geom_contour_filled(aes(fill = z))`: Draw 2D contour plot with the space between lines filled. `aes()` arguments: `x`, `y`, `alpha`, `color`, `fill`, `group`, `linetype`, `linewidth`, `subgroup`.
- `l + geom_raster(aes(fill = z), hjust = 0.5, vjust = 0.5, interpolate = FALSE)`: Draw a raster plot. `aes()` arguments: `x`, `y`, `alpha`, `fill`.
- `l + geom_tile(aes(fill = z))`: Draw a tile plot. `aes()` arguments: `x`, `y`, `alpha`, `color`, `fill`, `linetype`, `linewidth`, `width`.

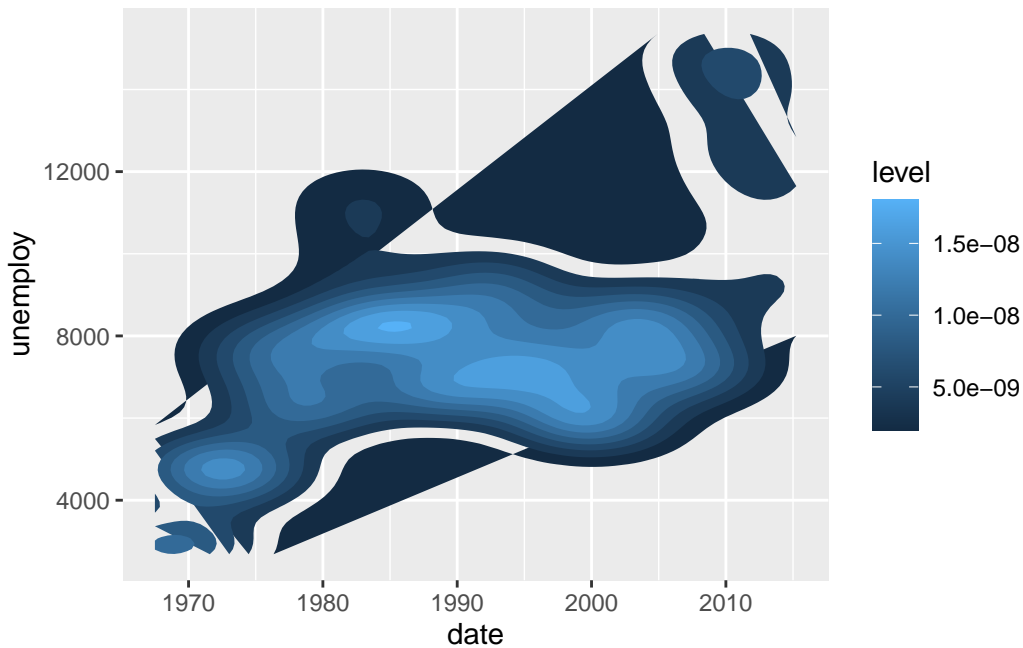
### Stats

An alternative way to build a layer.

A stat builds new variables to plot (e.g., count, prop).

Visualize a stat by changing the default stat of a geom function, `geom_bar(stat = "count")`, or by using a stat function, `stat_count(geom = "bar")`, which calls a default geom to make a layer (equivalent to a geom function). Use `after_stat(name)` syntax to map the stat variable `name` to an aesthetic.

```
i + stat_density_2d(aes(fill = after_stat(level)), geom = "polygon")
```



In this example, "polygon" is the geom to use, `stat_density_2d()` is the stat function, `aes()` contains the geom mappings, and `level` is the variable created by stat.

- `c + stat_bin(binwidth = 1, boundary = 10): x, y | count, ncount, density, ndensity`
- `c + stat_count(width = 1): x, y | count, density`
- `c + stat_density(adjust = 1, kernel = "gaussian"): x, y | count, density, scaled`
- `e + stat_bin_2d(bins = 30, drop = T): x, y, fill | count, density`
- `e + stat_bin_hex(bins = 30): x, y, fill | count, density`
- `e + stat_density_2d(contour = TRUE, n = 100): x, y, color, linewidth | level`
- `e + stat_ellipse(level = 0.95, segments = 51, type = "t")`

- `l + stat_contour(aes(z = z)): x, y, z, order | level`
- `l + stat_summary_hex(aes(z = z), bins = 30, fun = max): x, y, z, fill | value`
- `l + stat_summary_2d(aes(z = z), bins = 30, fun = mean): x, y, z, fill | value`
- `f + stat_boxplot(coef = 1.5): x, y | lower, middle, upper, width, ymin, ymax`
- `f + stat_ydensity(kernel = "gaussian", scale = "area"): x, y | density, scaled, count, n, violinwidth, width`
- `e + stat_ecdf(n = 40): x, y | x, y`
- `e + stat_quantile(quantiles = c(0.1, 0.9), formula = y ~ log(x), method = "rq"): x, y | quantile`
- `e + stat_smooth(method = "lm", formula = y ~ x, se = T, level = 0.95): x, y | se, x, y, ymin, ymax`
- `ggplot() + xlim(-5, 5) + stat_function(fun = dnorm, n = 20, geom = "point"): x | x, y`
- `ggplot() + stat_qq(aes(sample = 1:100)): x, y, sample | sample, theoretical`
- `e + stat_sum(): x, y, size | n, prop`
- `e + stat_summary(fun.data = "mean_cl_boot")`
- `h + stat_summary_bin(fun = "mean", geom = "bar")`
- `e + stat_identity()`
- `e + stat_unique()`

## Scales

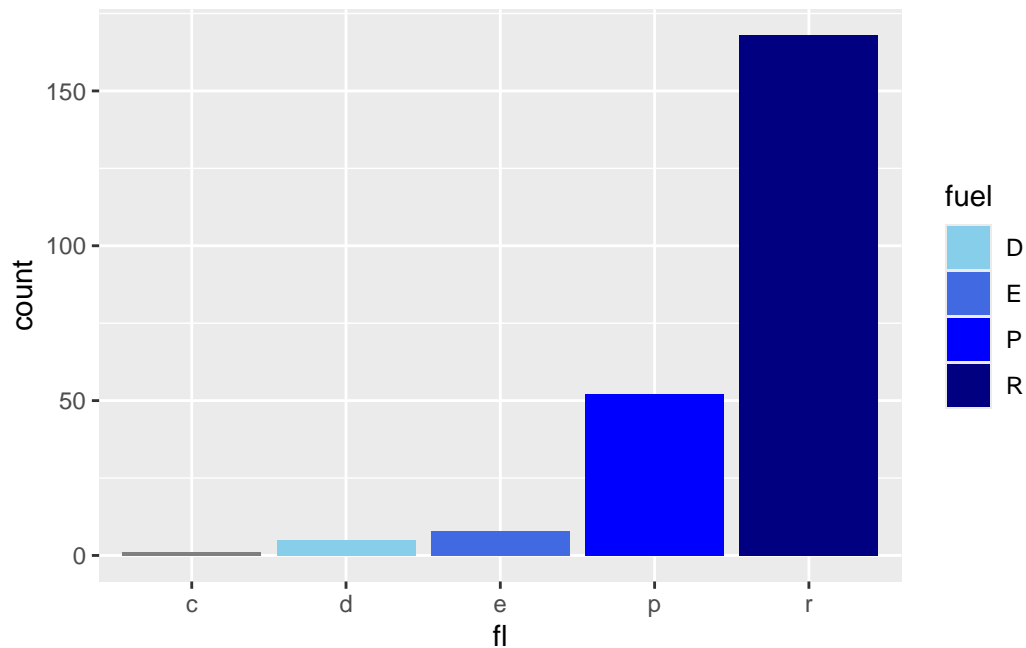
Override defaults with **scales** package.

**Scales** map data values to the visual values of an aesthetic. To change a mapping, add a new scale.

```
n <- d + geom_bar(aes(fill = fl))

n + scale_fill_manual(
  values = c("skyblue", "royalblue", "blue", "navy"),
  limits = c("d", "e", "p", "r"),
  breaks = c("d", "e", "p", "r"),
  name = "fuel",
```

```
labels = c("D", "E", "P", "R")
)
```



In this example, `scale_` specifies a scale function, `fill` is the aesthetic to adjust, and `manual` is the prepackaged scale to use.

`values` contains scale-specific arguments, `limits` specifies the range of values to include in mappings, `breaks` specifies the breaks to use in legend/axis, and `name` and `labels` specify the title and labels to use in the legend/axis.

## General Purpose Scales

Use with most aesthetics.

- `scale*_continuous()`: Map continuous values to visual ones.
- `scale*_discrete()`: Map discrete values to visual ones.
- `scale*_binned()`: Map continuous values to discrete bins.
- `scale*_identity()`: Use data values as visual ones.
- `scale*_manual(values = c())`: Map discrete values to manually chosen visual ones.

- `scale*_date(date_labels = "%m/%d", date_breaks = "2 weeks")`: Treat data values as dates.
- `scale*_datetime()`: Treat data values as date times. Same as `scale*_date()`. See `?strptime` for label formats.

## X & Y Location Scales

Use with x or y aesthetics (x shown here).

- `scale_x_log10()`: Plot x on log10 scale.
- `scale_x_reverse()`: Reverse the direction of the x axis.
- `scale_x_sqrt()`: Plot x on square root scale.

## Color and Fill Scales (Discrete)

- `n + scale_fill_brewer(palette = "Blues")`: Use color scales from ColorBrewer. For palette choices `RColorBrewer::display.brewer.all()`.
- `n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")`: Use a grey gradient color scale.

## Color and Fill Scales (Continuous)

```
o <- c + geom_dotplot(aes(fill = ..x..))
```

- `o + scale_fill_distiller(palette = "Blues")`: Interpolate a palette into a continuous scale.
- `o + scale_fill_gradient(low = "red", high = "yellow")`: Create a two color gradient.
- `o + scale_fill_gradient2(low = "red", high = "blue", mid = "white", midpoint = 25)`: Create a diverging color gradient.
- `o + scale_fill_gradientn(colors = topo.colors(6))`: Create a n-color gradient. Also `rainbow()`, `heat.colors()`, `terrain.colors()`, `cm.colors()`, `RColorBrewer::brewer.pal()`.

## Shape and Size Scales

```
p <- e + geom_point(aes(shape = fl, size = cyl))
```

- `p + scale_shape() + scale_size()`: Map discrete values to shape and size aesthetics.
- `p + scale_shape_manual(values = c(3:7))`: Map discrete values to specified shape values.
- `p + scale_radius(range = c(1,6))`: Map values to a shape's radius.
- `p + scale_size_area(max_size = 6)`: Like `scale_size()` but maps zero values to zero size.

Shapes used here are the same as the ones listed in the Aes section.

## Coordinate Systems

```
u <- d + geom_bar()
```

- `u + coord_cartesian(xlim = c(0, 5))`: `xlim`, `ylim`. The default Cartesian coordinate system.
- `u + coord_fixed(ratio = 1/2)`: `ratio`, `xlim`, `ylim`. Cartesian coordinates with fixed aspect ratio between x and y units.
- `ggplot(mpg, aes(y = fl)) + geom_bar()`: Flip Cartesian coordinates by switching x and y aesthetic mappings.
- `u + coord_polar(theta = "x", direction = 1)`: `theta`, `start`, `direction`. Polar coordinates.
- `u + coord_trans(y = "sqrt")`: `x`, `y`, `xlim`, `ylim`. Transformed Cartesian coordinates. Set `xtrans` and `ytrans` to the name of a window function.
- `+ coord_sf()`: `xlim`, `ylim`, `crs`. Ensures all layers use a common Coordinate Reference System.

## Position Adjustments

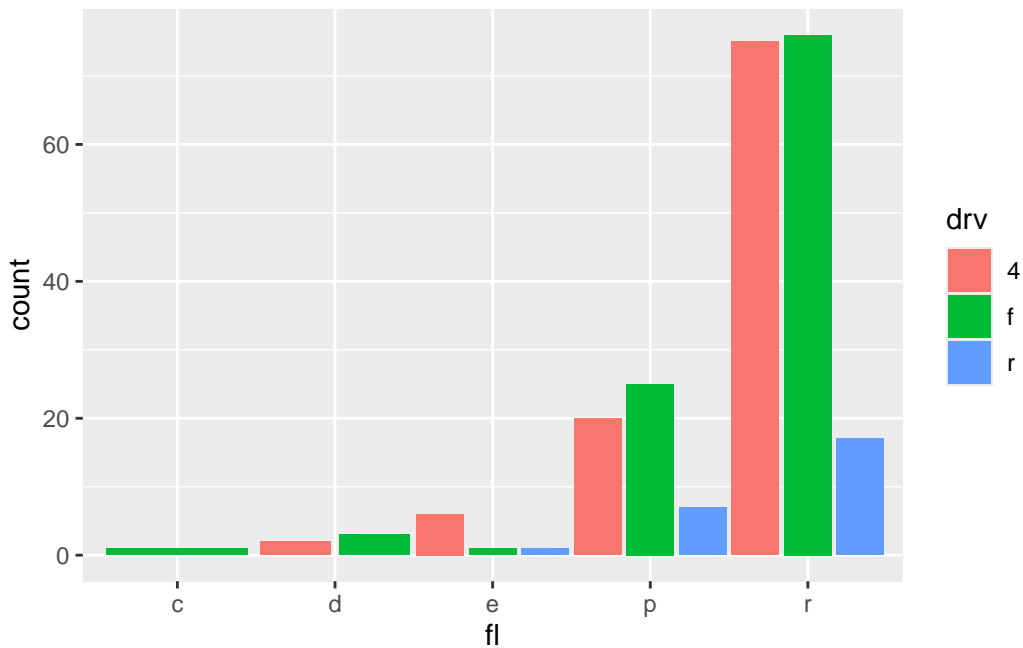
Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

```
s <- ggplot(mpg, aes(fl, fill = drv))
```

- `s + geom_bar(position = "dodge")`: Arrange elements side by side.
- `s + geom_bar(position = "fill")`: Stack elements on top of one another, normalize height.
- `e + geom_point(position = "jitter")`: Add random noise to X and Y position of each element to avoid overplotting.
- `e + geom_label(position = "nudge")`: Nudge labels away from points.
- `s + geom_bar(position = "stack")`: Stack elements on top of one another.

Each position adjustment can be recast as a function with manual `width` and `height` arguments:

```
s + geom_bar(position = position_dodge(width = 1))
```

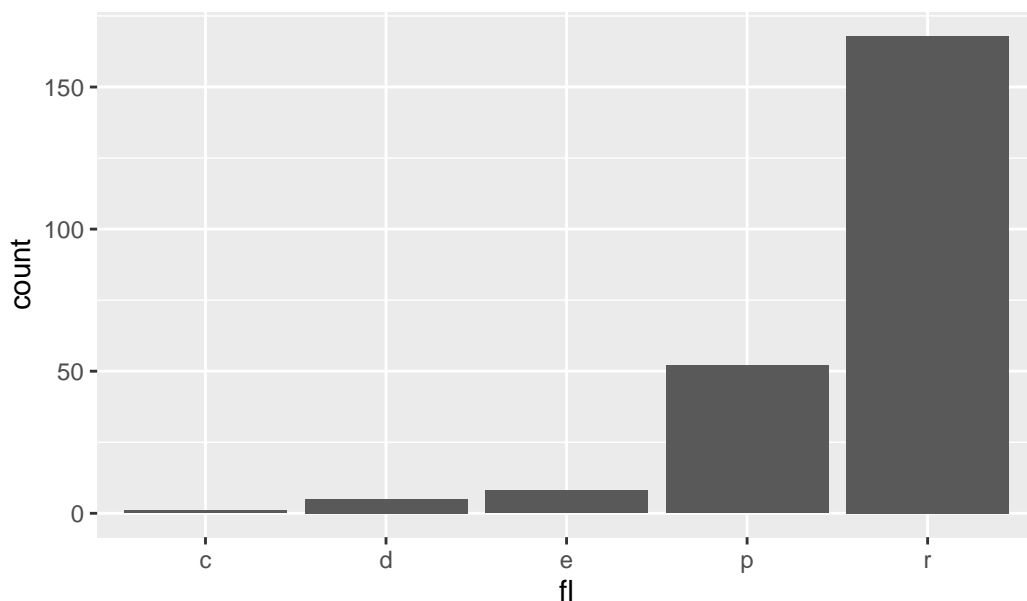


## Themes

- `u + theme_bw()`: White background with grid lines.
- `u + theme_gray()`: Grey background with white grid lines (default theme).
- `u + theme_dark()`: Dark grey background and grid lines for contrast.
- `u + theme_classic()`: No grid lines.
- `u + theme_light()`: Light grey axes and grid lines.
- `u + theme_linedraw()`: Uses only black lines.
- `u + theme_minimal()`: Minimal theme.
- `u + theme_void()`: Empty theme.
- `u + theme()`: Customize aspects of the theme such as axis, legend, panel, and facet properties.

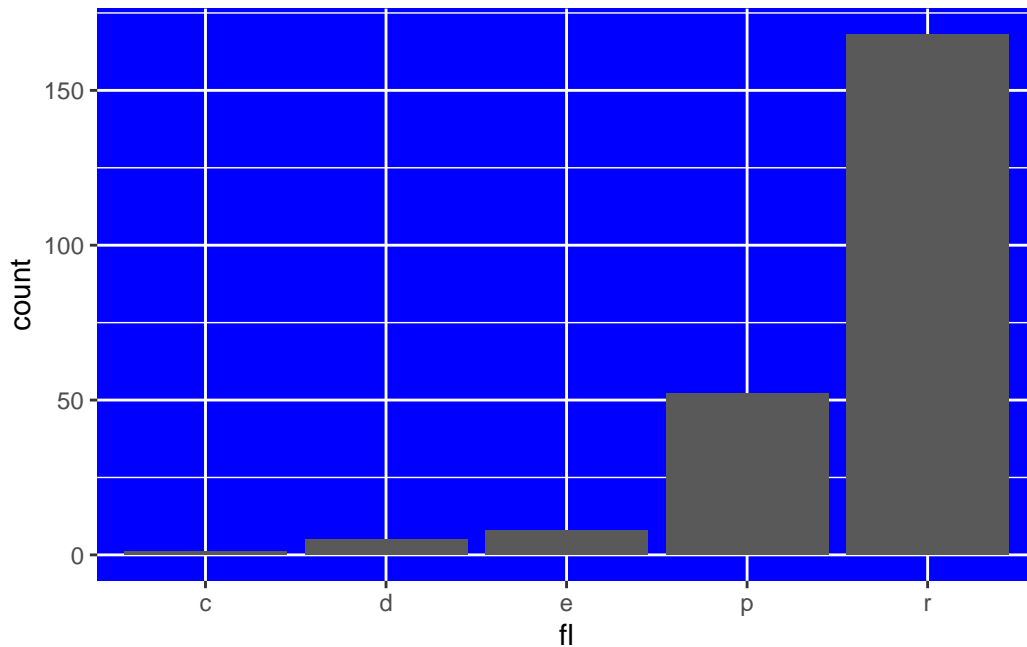
```
u + labs(title = "Title") + theme(plot.title.position = "plot")
```

### Title



```
u + theme(panel.background = element_rect(fill = "blue"))
```





## Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

```
t <- ggplot(mpg, aes(cty, hwy)) + geom_point()
```

- `t + facet_grid(. ~ fl)`: Facet into a column based on `fl`.
- `t + facet_grid(year ~ .)`: Facet into rows based on `year`.
- `t + facet_grid(year ~ fl)`: Facet into both rows and columns.
- `t + facet_wrap(~ fl)`: Wrap facets into a rectangular layout.
- `t + facet_grid(drv ~ fl, scales = "free")`: Set **scales** to let axis limits vary across facets. Also **"free\_x"** for x axis limits adjust to individual facets and **"free\_y"** for y axis limits adjust to individual facets.

Set **labeller** to adjust facet label:

- `t + facet_grid(. ~ fl, labeller = label_both)`: Labels each facet as “fl: c”, “fl: d”, etc.
- `t + facet_grid(fl ~ ., labeller = label_bquote(alpha ^ .(fl)))`: Labels each facet as “c”, “d”, etc.

## Labels and Legends

Use `labs()` to label elements of your plot.

```
t + labs(x = "New x axis label",
  y = "New y axis label",
  title = "Add a title above the plot",
  subtitle = "Add a subtitle below title",
  caption = "Add a caption below plot",
  alt = "Add alt text to the plot",
  <Aes> = "New <Aes> legend title")
```

- `t + annotate(geom = "text", x = 8, y = 9, label = "A")`: Places a geom with manually selected aesthetics.
- `p + guides(x = guide_axis(n.dodge = 2))`: Avoid crowded or overlapping labels with `guide_axis(n.dodge` or `angle`).
- `n + guides(fill = "none")`: Set legend type for each aesthetic: `colorbar`, `legend`, or `none` (no legend).
- `n + theme(legend.position = "bottom")`: Place legend at “bottom”, “top”, “left”, or “right”.
- `n + scale_fill_discrete(name = "Title", labels = c("A", "B", "C", "D", "E"))`: Set legend title and labels with a scale function.

## Zooming

- `t + coord_cartesian(xlim = c(0, 100), ylim = c(10,20))`: Zoom without clipping (preferred).
- `t + xlim(0, 100) + ylim(10, 20)` or `t + scale_x_continuous(limits = c(0, 100)) + scale_y_continuous(limits = c(0, 100))`: Zoom with clipping (removes unseen data points).

---

CC BY SA Posit Software, PBC • [info@posit.co](mailto:info@posit.co) • [posit.co](https://posit.co)

Learn more at [ggplot2.tidyverse.org](https://ggplot2.tidyverse.org).

Updated: 2025-03.

```
packageVersion("ggplot2")
```

```
[1] '3.5.1'
```

---