Data Visualization

Matthew McDonald

Loading the Tidyverse

One line of code loads the core tidyverse; packages which you will use in almost every data analysis.

It also tells you which functions from the tidyverse conflict with functions in base R (or from other packages you might have loaded).

```r
library(tidyverse)
```

Explicit Calls to packages

If we need to be explicit about where a function (or dataset) comes from, we'll use the special form `package::function()`.

For example, `ggplot2::ggplot()` tells you explicitly that we're using the `ggplot()` function from the ggplot2 package.

First steps

First Question: Do cars with big engines use more fuel than cars with small engines?

You probably already have an answer, but try to make your answer precise.

- What does the relationship between engine size and fuel efficiency look like?
- Is it positive?
- Negative?
- Linear?
- Nonlinear?

The `mpg` data frame

You can test your answer with the `mpg` **data frame** found in ggplot2 (a.k.a. `ggplot2::mpg`).

A data frame is a rectangular collection of variables (in the columns) and observations (in the rows).

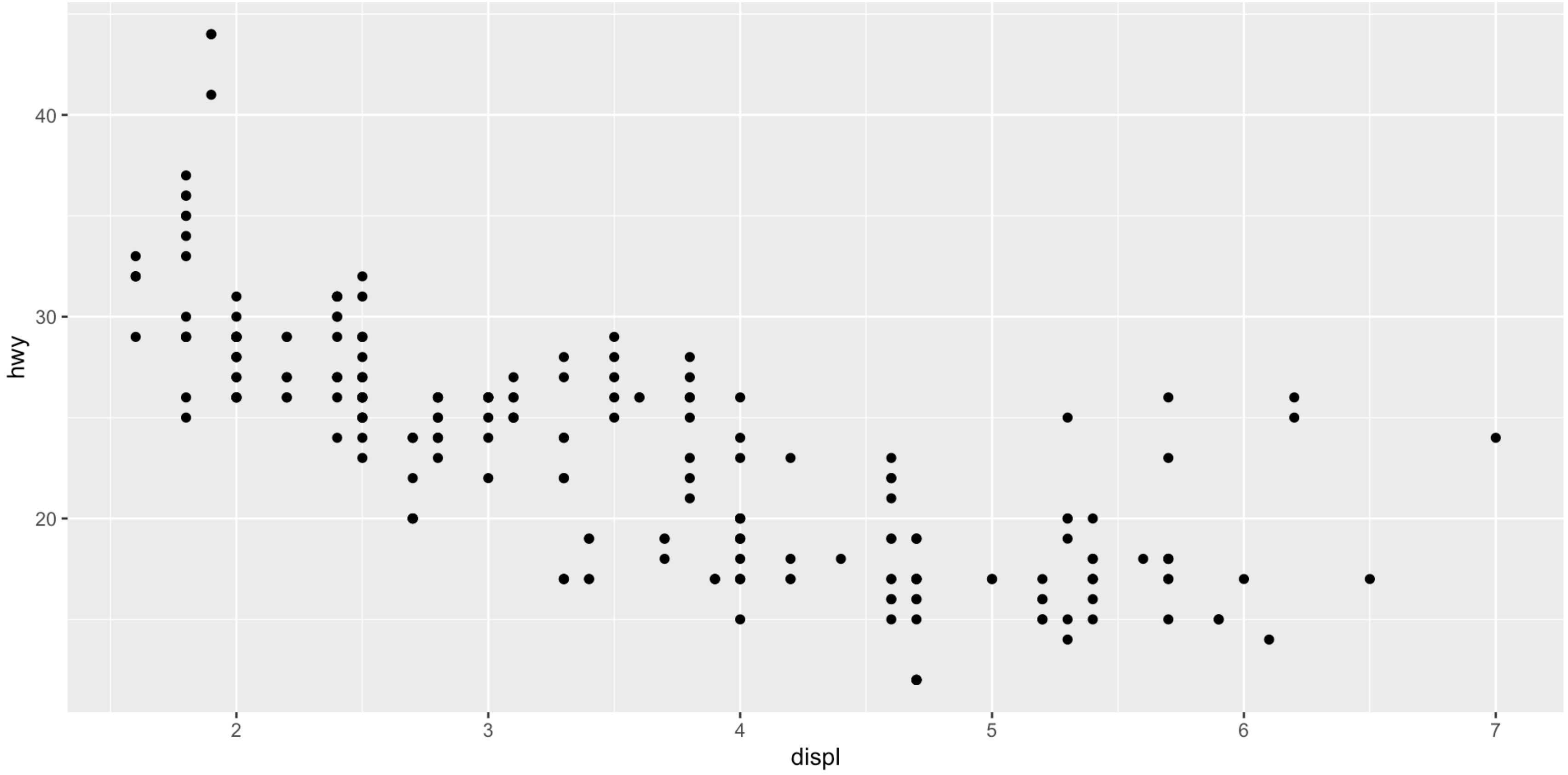
`mpg` contains observations collected by the US Environmental Protection Agency on 38 models of car.

```
mpg
```

```
# A tibble: 234 × 11
   manufacturer model      displ  year   cyl trans drv     cty   hwy fl    class
   <chr>        <chr>      <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
 1 audi         a4           1.8  1999     4 auto… f        18    29 p     comp…
 2 audi         a4           1.8  1999     4 manu… f        21    29 p     comp…
 3 audi         a4           2    2008     4 manu… f        20    31 p     comp…
 4 audi         a4           2    2008     4 auto… f        21    30 p     comp…
 5 audi         a4           2.8  1999     6 auto… f        16    26 p     comp…
 6 audi         a4           2.8  1999     6 manu… f        18    26 p     comp…
 7 audi         a4           3.1  2008     6 auto… f        18    27 p     comp…
 8 audi         a4 quattro   1.8  1999     4 manu… 4        18    26 p     comp…
 9 audi         a4 quattro   1.8  1999     4 auto… 4        16    25 p     comp…
10 audi         a4 quattro   2    2008     4 manu… 4        20    28 p     comp…
# ℹ 224 more rows
```

Creating a ggplot

```r
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```



ggplot call

With ggplot2, you begin a plot with the function `ggplot()`.

- `ggplot()` creates a coordinate system that you can add layers to.
- The first argument of `ggplot()` is the dataset to use in the graph.
- `ggplot(data = mpg)` creates an empty graph

Complete your graph by adding one or more layers to `ggplot()`.

- The function `geom_point()` adds a layer of points to your plot, which creates a scatterplot.

mapping argument

Each geom function in ggplot2 takes a `mapping` argument.

- This defines how variables in your dataset are mapped to visual properties of your plot.

The `mapping` argument is always paired with `aes()`, and the `x` and `y` arguments of `aes()` specify which variables to map to the x and y axes.

- ggplot2 looks for the mapped variables in the `data` argument, in this case, `mpg`.

A graphing template

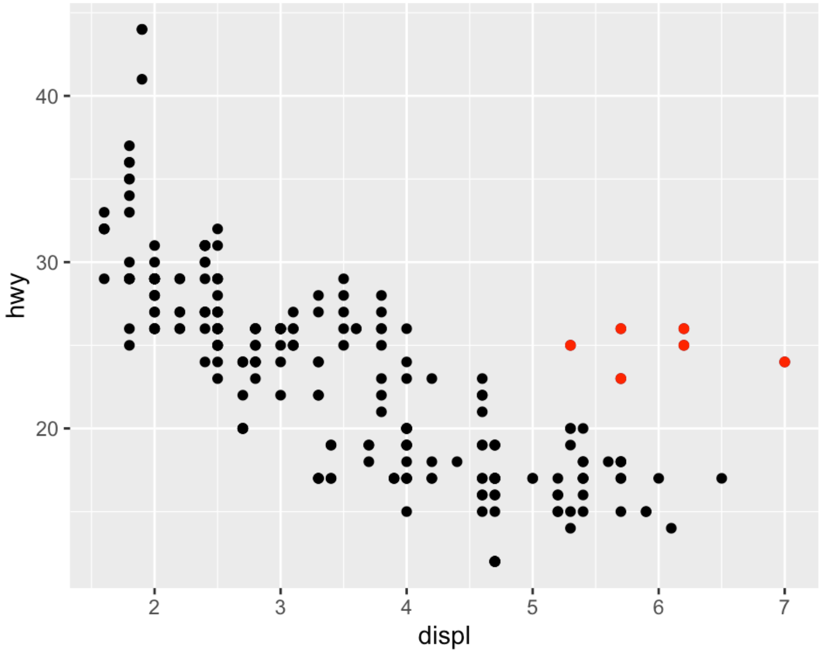
```r
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

Aesthetic mappings (1)

In this plot, one group of points (highlighted in red) seems to fall outside of the linear trend.

These cars have a higher mileage than you might expect.

How can you explain these cars?

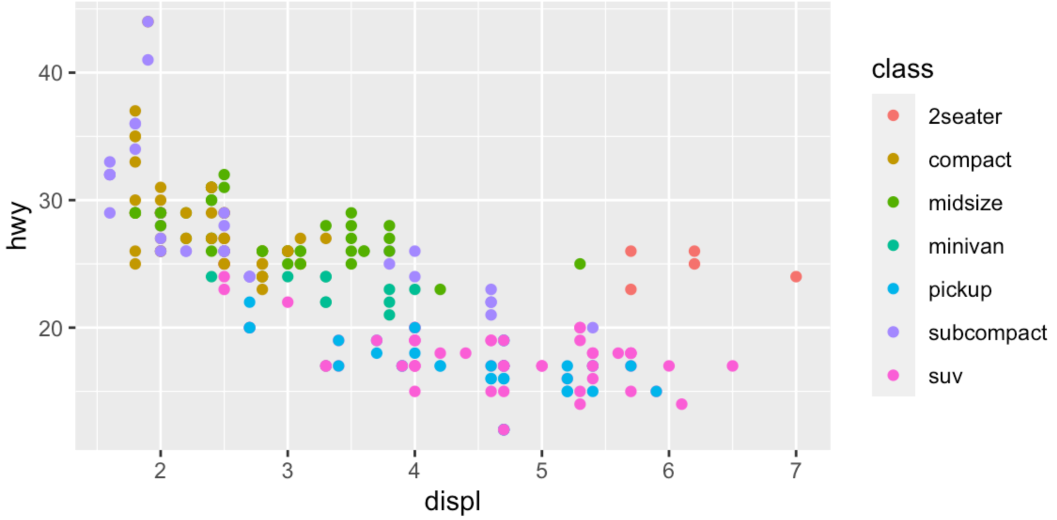


Aesthetic mappings (2)

You can add a third variable, like `class`, to a two dimensional scatterplot by mapping it to an **aesthetic**.

An aesthetic is a visual property of the objects in your plot.

```r
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



Facets

Another way, particularly useful for categorical (aka factor) variables, is to split your plot into **facets**, subplots that each display one subset of the data.

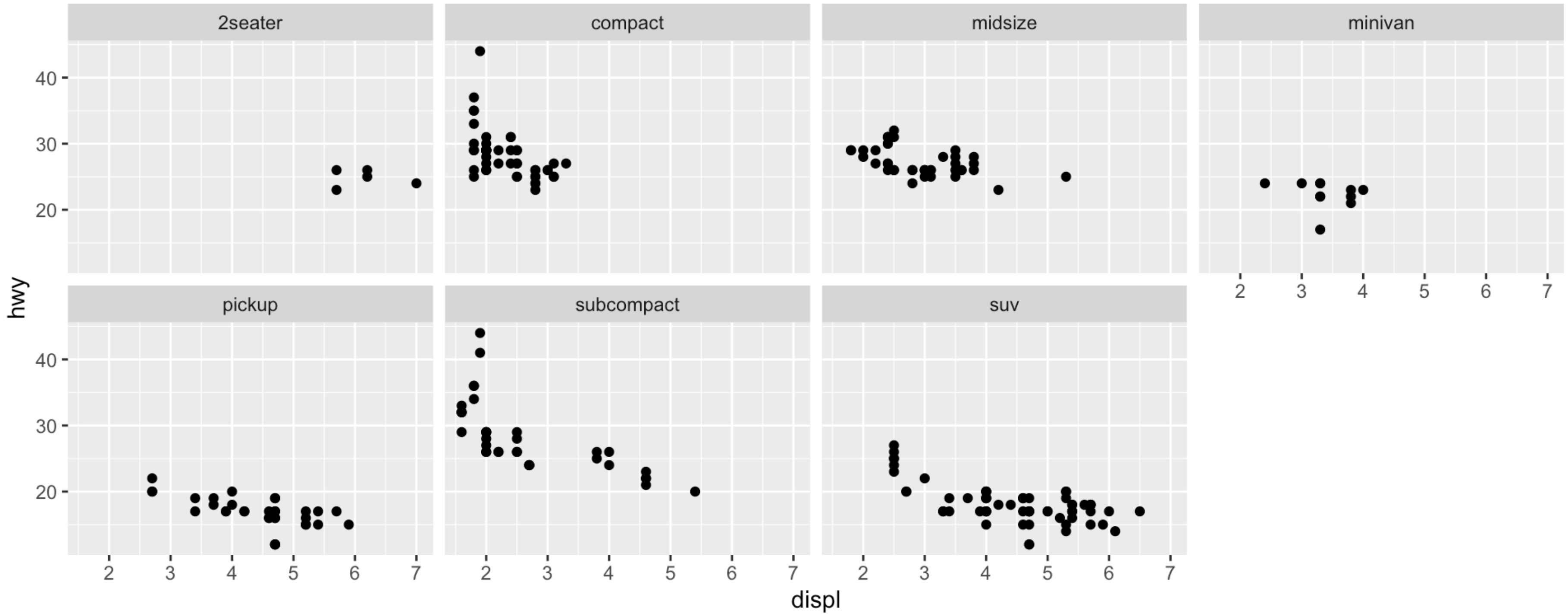
To facet your plot by a single variable, use `facet_wrap()`.

The first argument of `facet_wrap()` is a formula, which you create with `~` followed by a variable name (here, “formula” is the name of a data structure in R, not a synonym for “equation”).

The variable that you pass to `facet_wrap()` should be discrete.

facet_wrap()

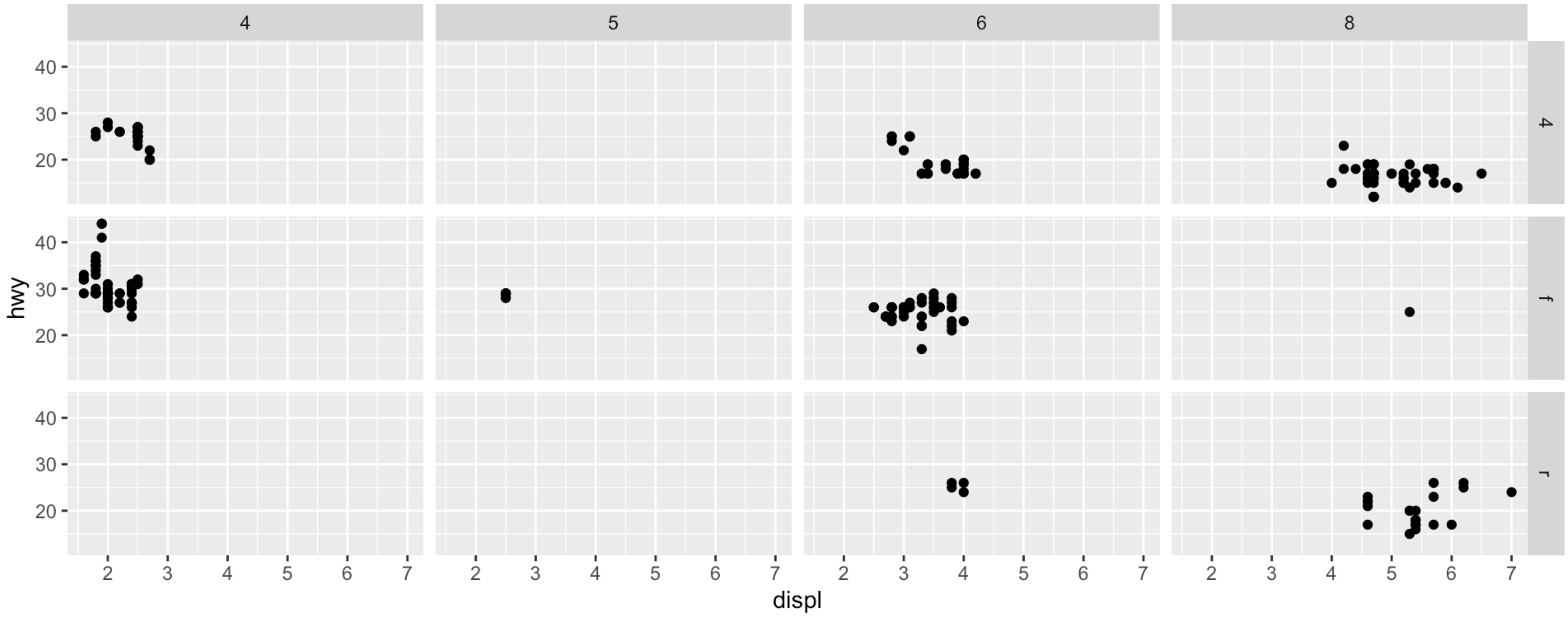
```r
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~ class, nrow = 2)
```



facet_grid()

To facet your plot on the combination of two variables, add `facet_grid()` to your plot call. The first argument of `facet_grid()` is also a formula containing two variable names separated by a `~`.

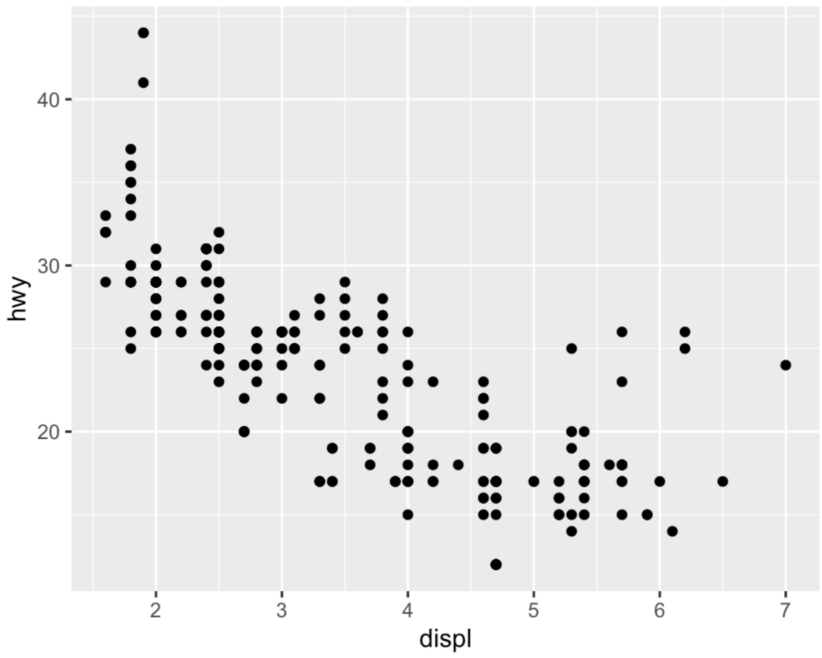
```r
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_grid(drv ~ cyl)
```

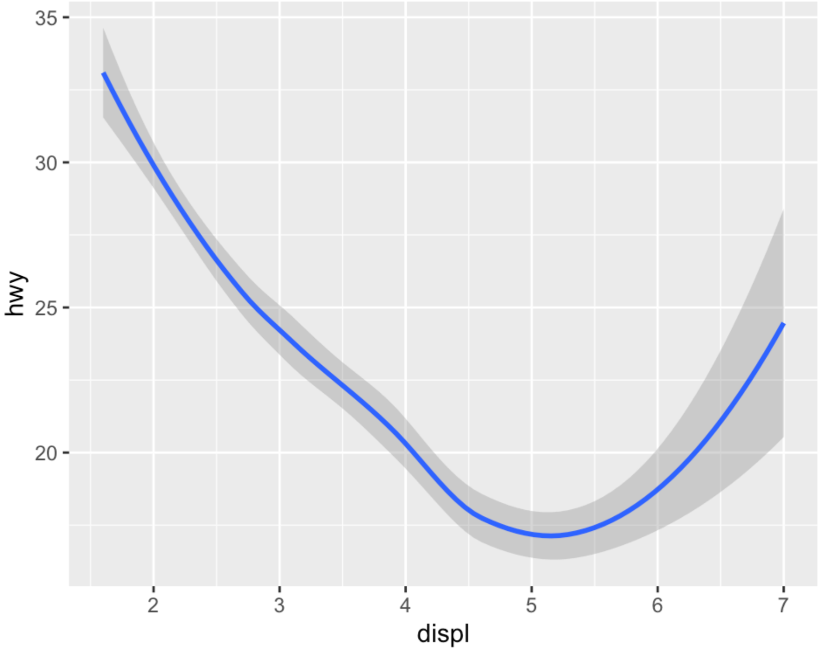


How are these two plots similar?

```r
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```

```r
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy))
```





Geometric Objects

Each plot uses a different visual object to represent the data. In ggplot2 syntax, we say that they use different **geoms**.

A **geom** is the geometrical object that a plot uses to represent data.

For example, bar charts use bar geoms, line charts use line geoms, boxplots use boxplot geoms, and so on. Scatterplots break the trend; they use the point geom.

Every geom function in ggplot2 takes a `mapping` argument. However, not every aesthetic works with every geom. You could set the shape of a point, but you couldn't set the "shape" of a line. On the other hand, you *could* set the linetype of a line.

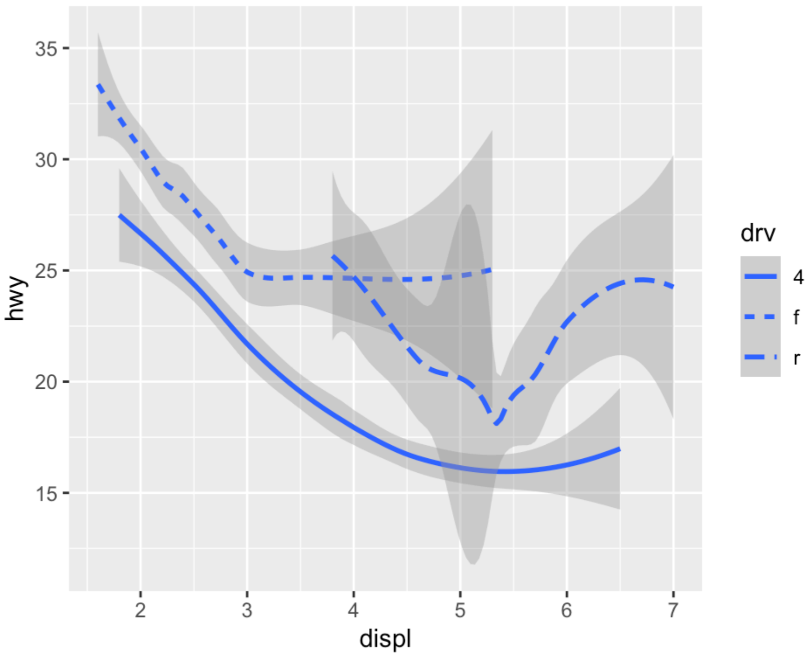
geom_smooth()

geom_smooth() will draw a different line, with a different linetype, for each unique value of the variable that you map to linetype.

Here geom_smooth() separates the cars into three lines based on their drv value, which describes a car's drive train.

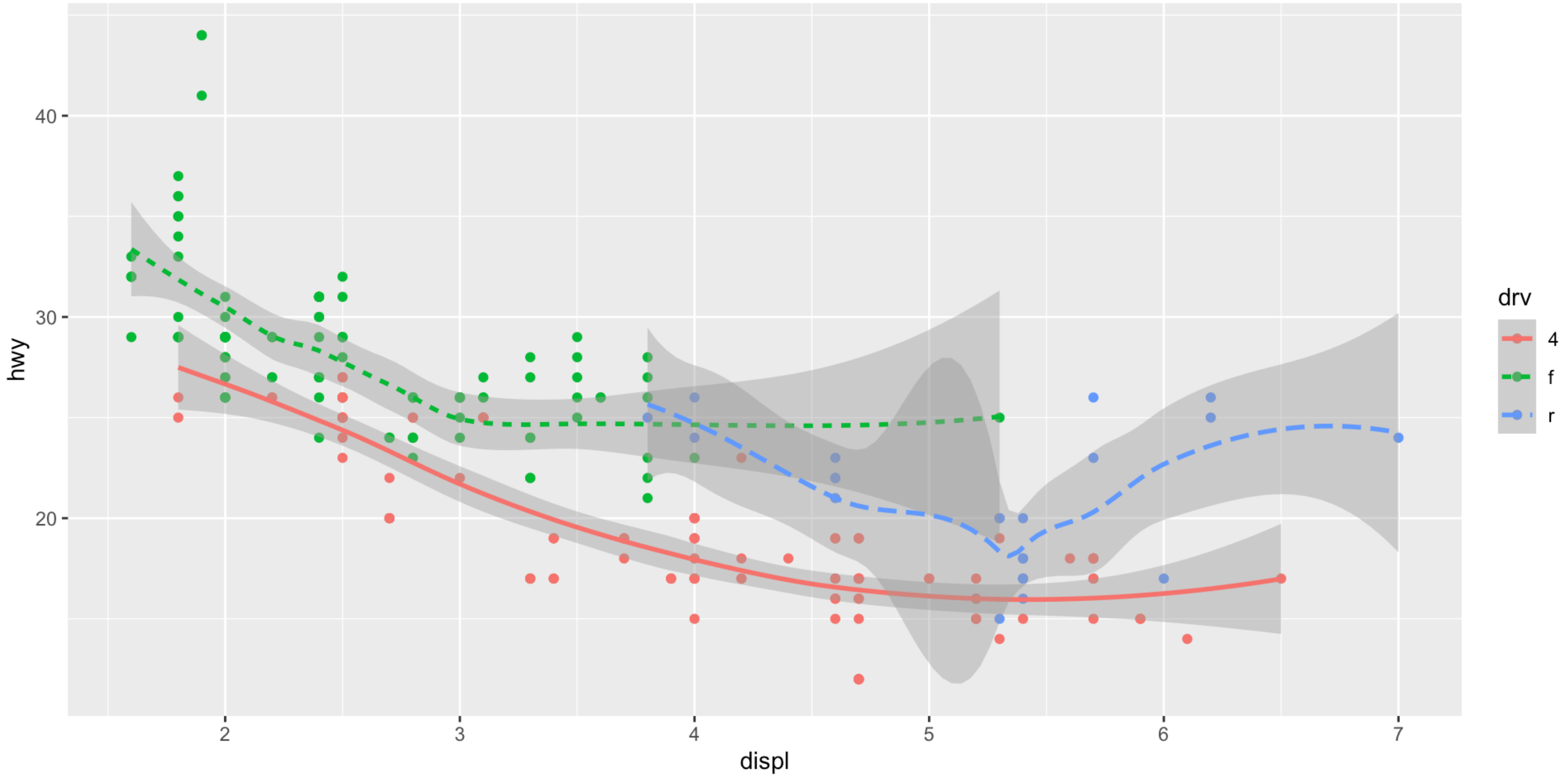
Here, 4 stands for four-wheel drive, f for front-wheel drive, and r for rear-wheel drive.

```
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ,
                            y = hwy,
                            linetype = drv))
```



Coloring by drive

If this sounds strange, we can make it more clear by overlaying the lines on top of the raw data and then colouring everything according to `drv`.



More Geoms

ggplot2 provides over 40 geoms, and extension packages provide even more (see https://exts.ggplot2.tidyverse.org/gallery/ for a sampling).

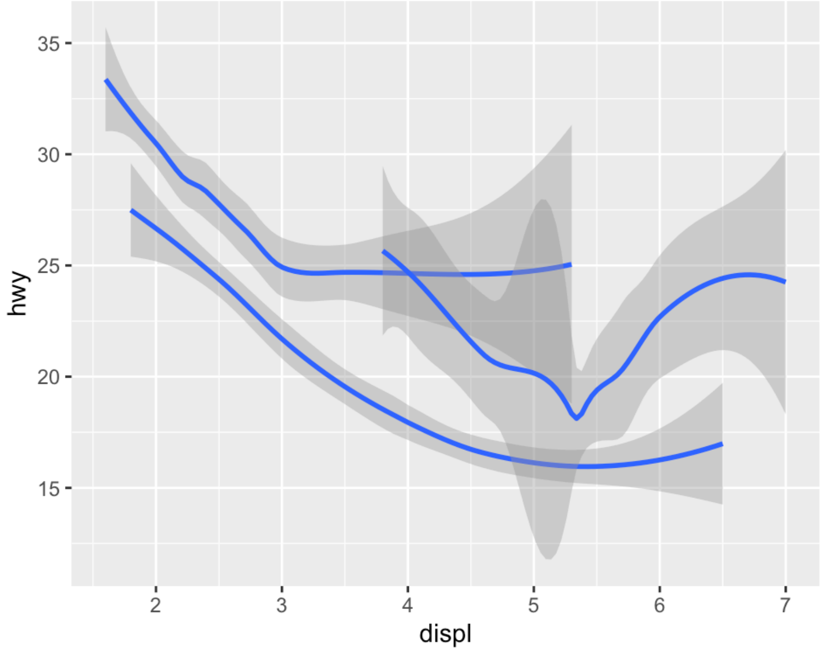
The best way to get a comprehensive overview is the ggplot2 cheatsheet, which you can find at http://rstudio.com/resources/cheatsheets.

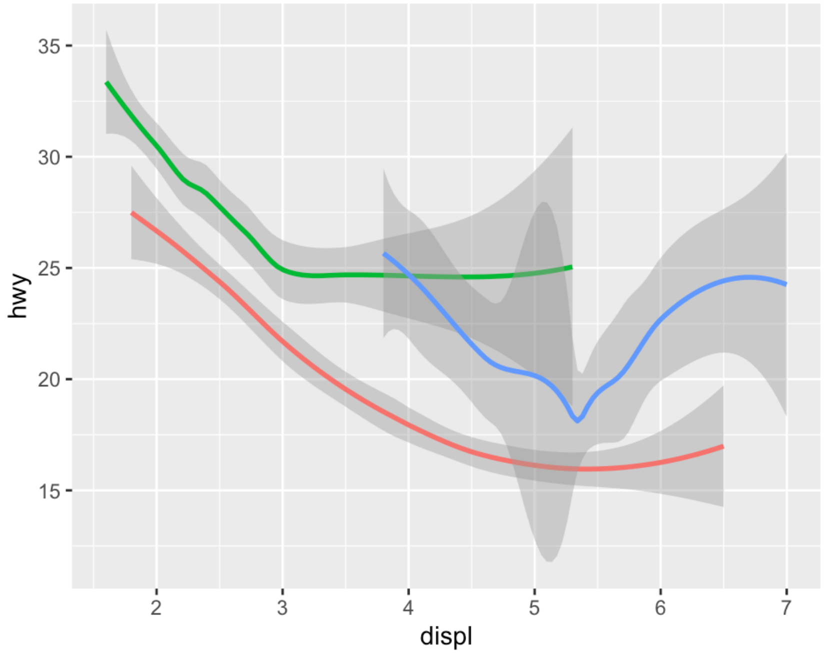
To learn more about any single geom, use help, e.g. `?geom_smooth`.

Grouping

```r
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ,
                            y = hwy,
                            group = drv))
```

```r
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ,
                            y = hwy,
                            color = drv),
      show.legend = FALSE)
```

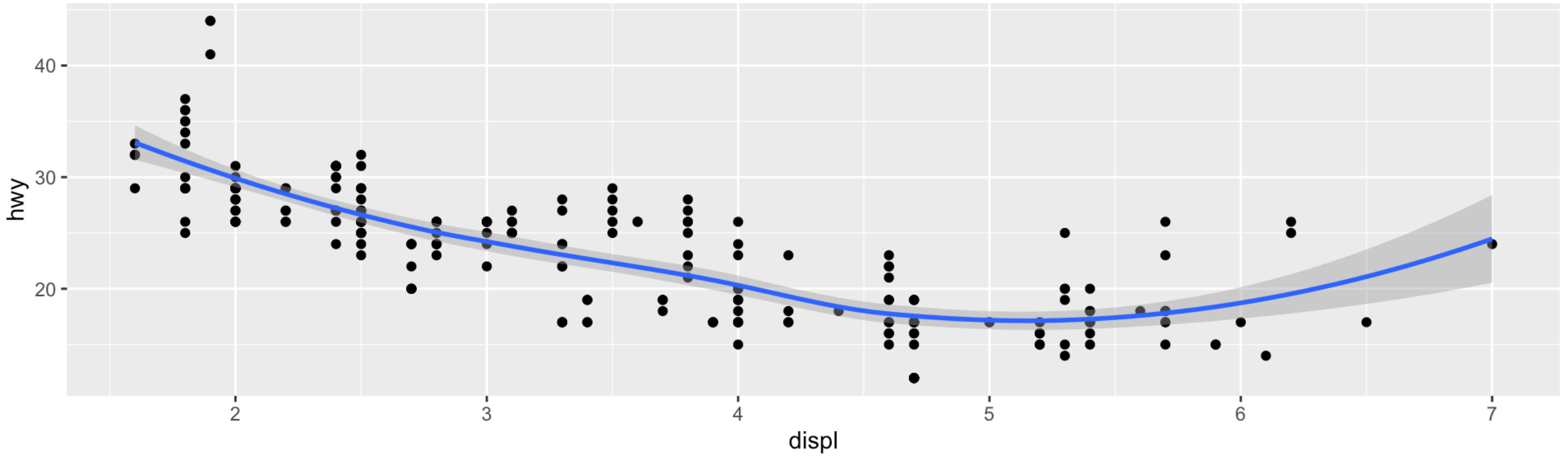




Multiple Geoms in Same Plot

To display multiple geoms in the same plot, add multiple geom functions to `ggplot()`:

```r
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  geom_smooth(mapping = aes(x = displ, y = hwy))
```



This, however, introduces some duplication in our code.

Global Mappings

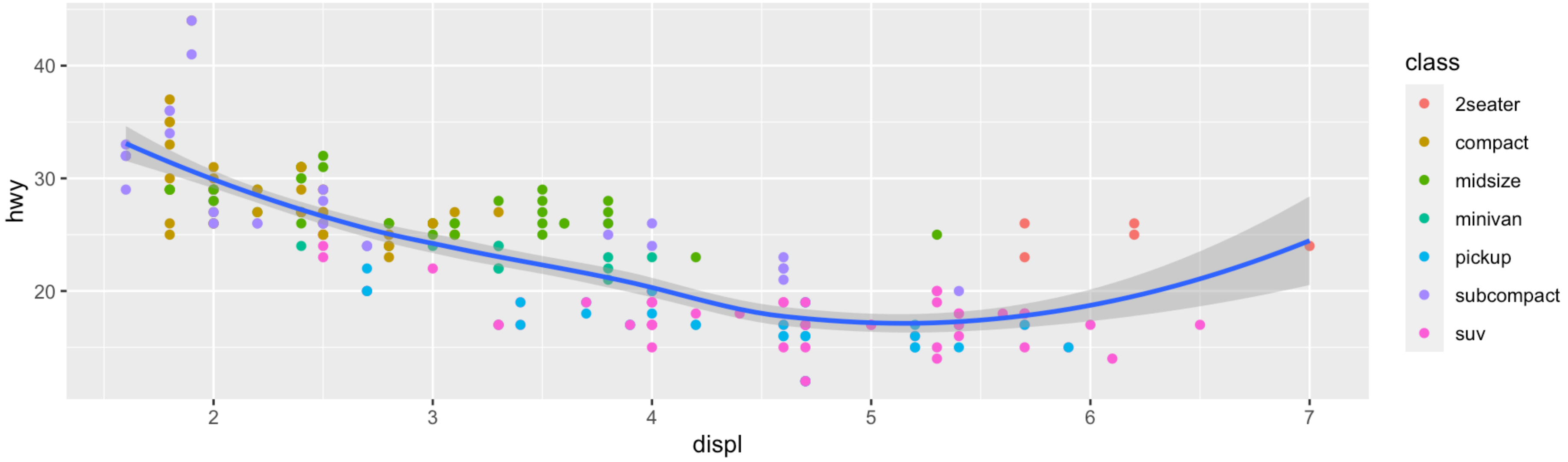
You can avoid this type of repetition by passing a set of mappings to `ggplot()`. ggplot2 will treat these mappings as global mappings that apply to each geom in the graph. In other words, this code will produce the same plot as the previous code:

```r
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth()
```

Layer-Specific Mappings

If you place mappings in a geom function, ggplot2 will treat them as local mappings for the layer. It will use these mappings to extend or overwrite the global mappings *for that layer only*. This makes it possible to display different aesthetics in different layers.

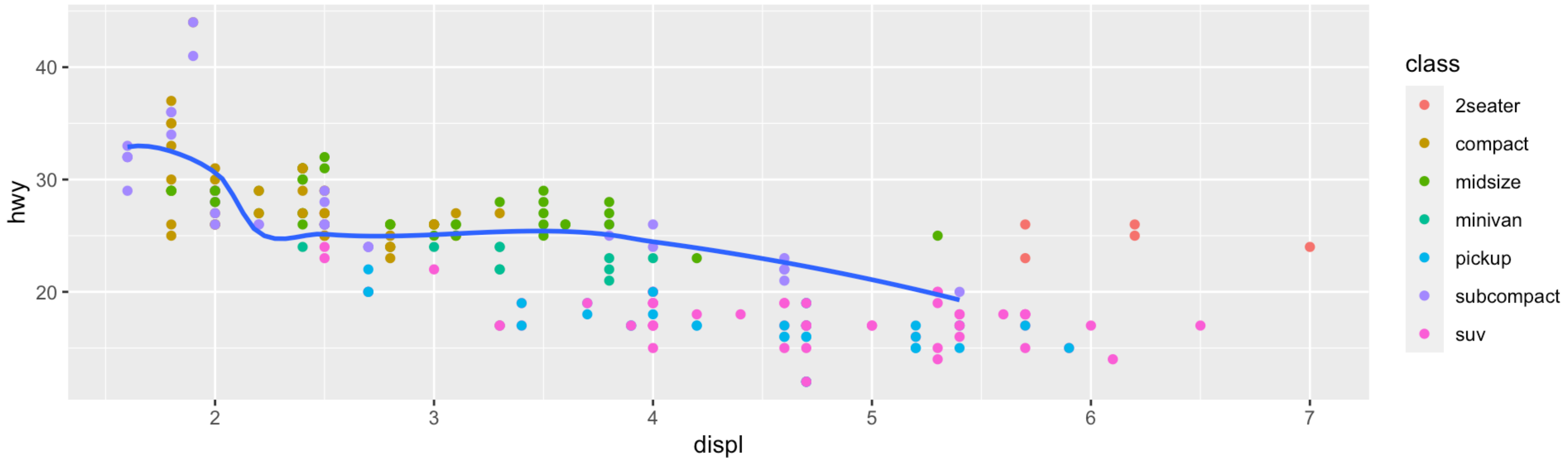
```r
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(color = class)) +
  geom_smooth()
```



Layer-Specific Data

You can use the same idea to specify different `data` for each layer. Here, our smooth line displays just a subset of the `mpg` dataset, the subcompact cars. The local data argument in `geom_smooth()` overrides the global data argument in `ggplot()` for that layer only.

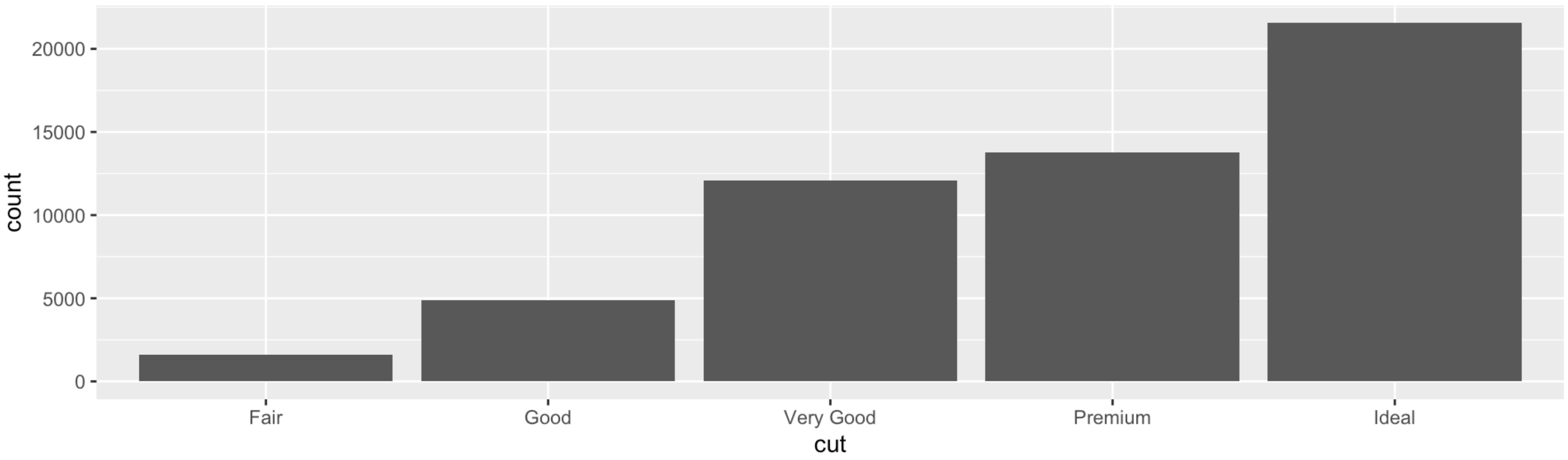
```r
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(color = class)) +
  geom_smooth(data = filter(mpg, class == "subcompact"), se = FALSE)
```



Bar Charts

The following chart displays the total number of diamonds in the `diamonds` dataset (a dataset included in ggplot2), grouped by `cut`. The chart shows that more diamonds are available with high quality cuts than with low quality cuts.

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut))
```



Bar Charts (2)

On the x-axis, the chart displays `cut`, a variable from `diamonds`.

On the y-axis, it displays count, but count is not a variable in `diamonds`!

Where does count come from?

Many graphs, like scatterplots, plot the raw values of your dataset.

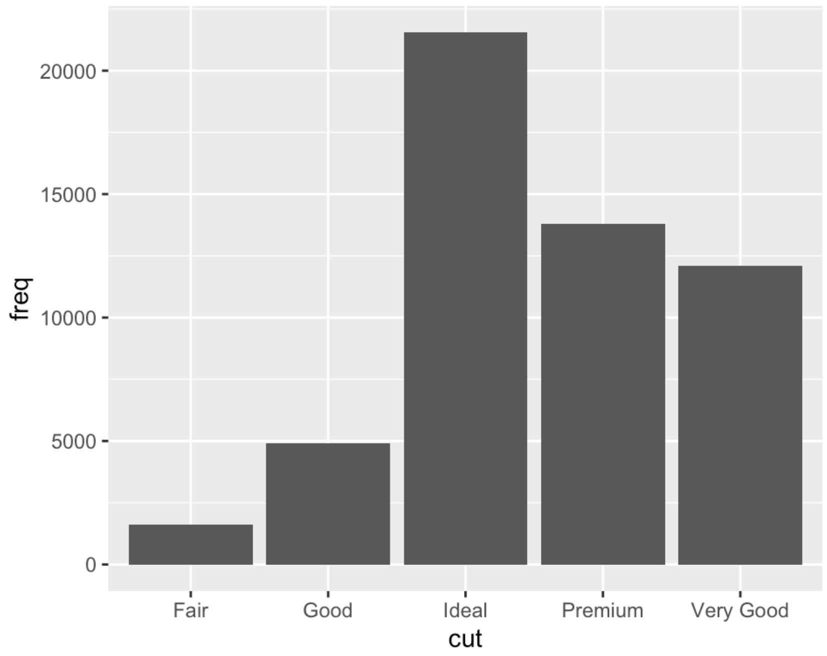
Other graphs, like bar charts, calculate new values to plot:

- bar charts, histograms, and frequency polygons bin your data and then plot bin counts, the number of points that fall in each bin.
- smoothers fit a model to your data and then plot predictions from the model.
- boxplots compute a robust summary of the distribution and then display a specially formatted box.

What if my data does include count data?

```r
demo <- tribble(
  ~cut,          ~freq,
  "Fair",        1610,
  "Good",        4906,
  "Very Good",   12082,
  "Premium",     13791,
  "Ideal",       21551
)

ggplot(data = demo) +
  geom_bar(mapping = aes(x = cut,
                         y = freq),
           stat = "identity")
```

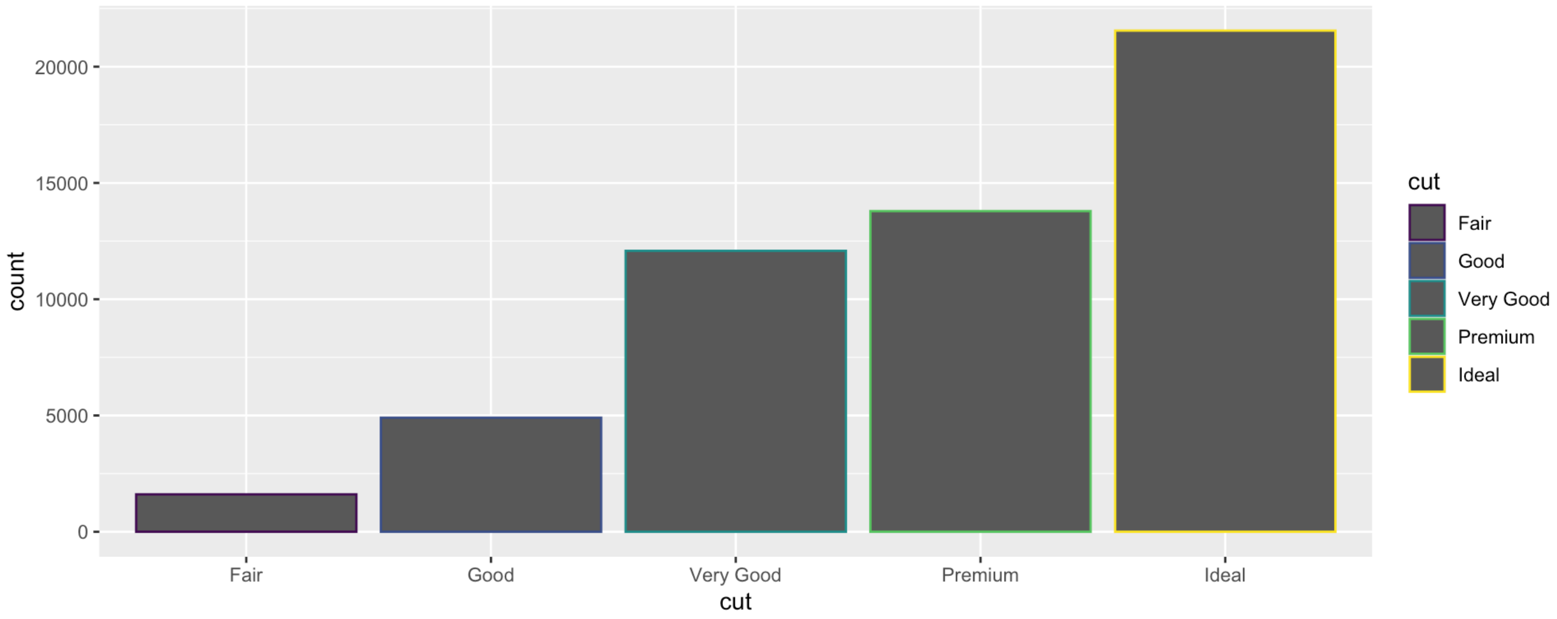


Color Aesthetic

There’s one more piece of magic associated with bar charts.

You can colour a bar chart using either the `color` aesthetic

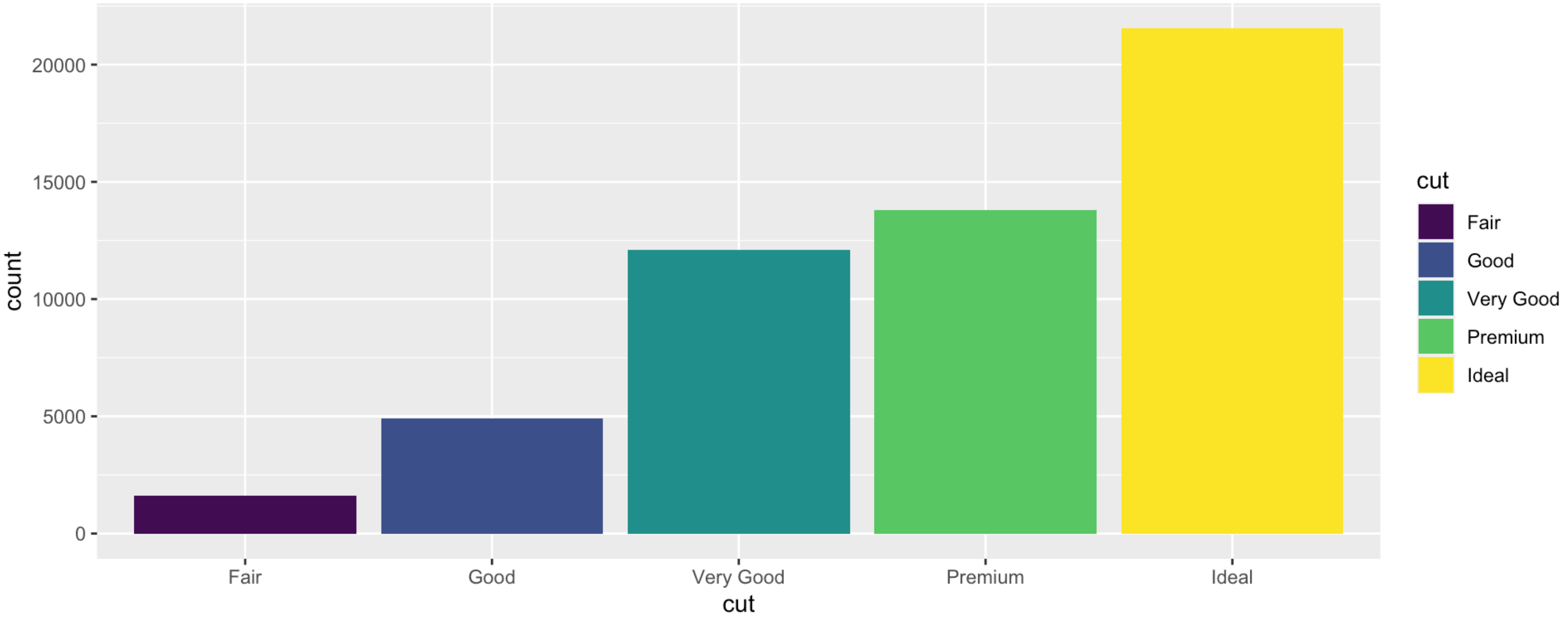
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, color = cut))
```



Fill Aesthetic

The Fill Aesthetic is more useful

```r
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = cut))
```

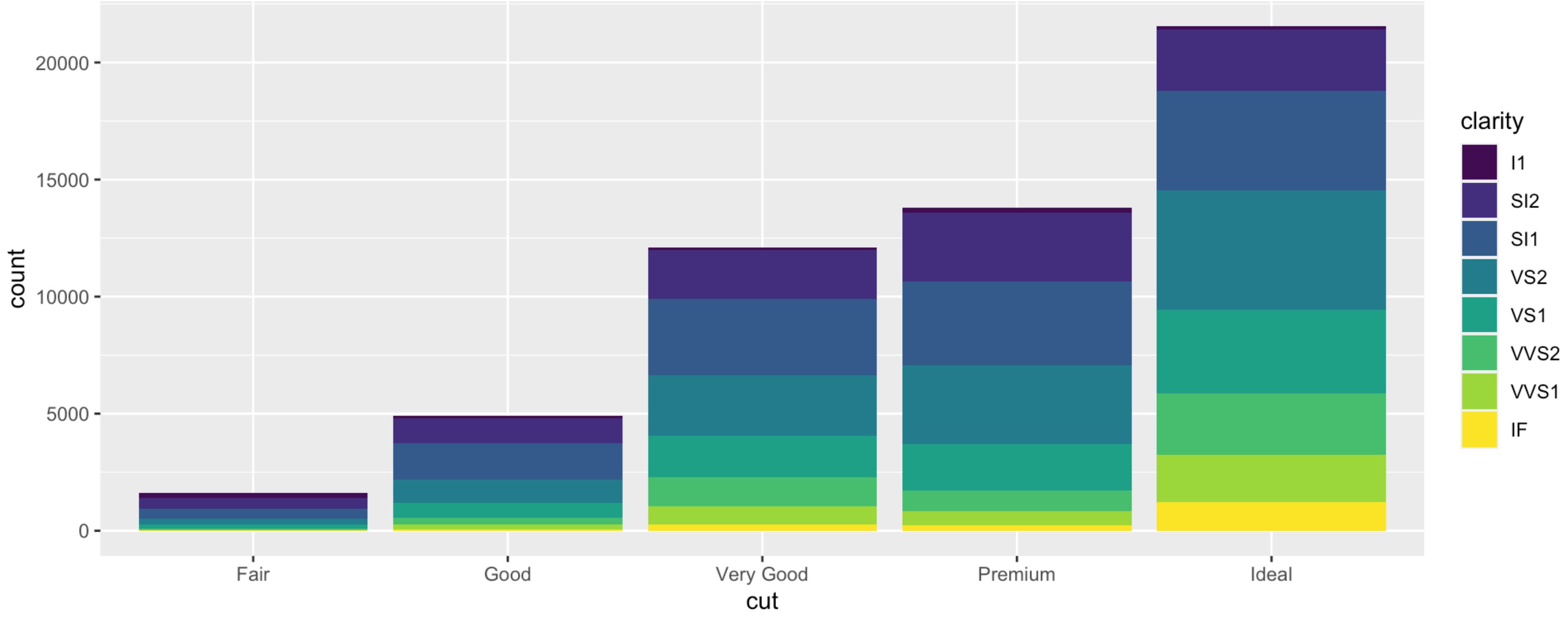


Stacking

Note what happens if you map the fill aesthetic to another variable, like `clarity`: the bars are automatically stacked.

Each colored rectangle represents a combination of `cut` and `clarity`.

```r
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity))
```



Position Argument

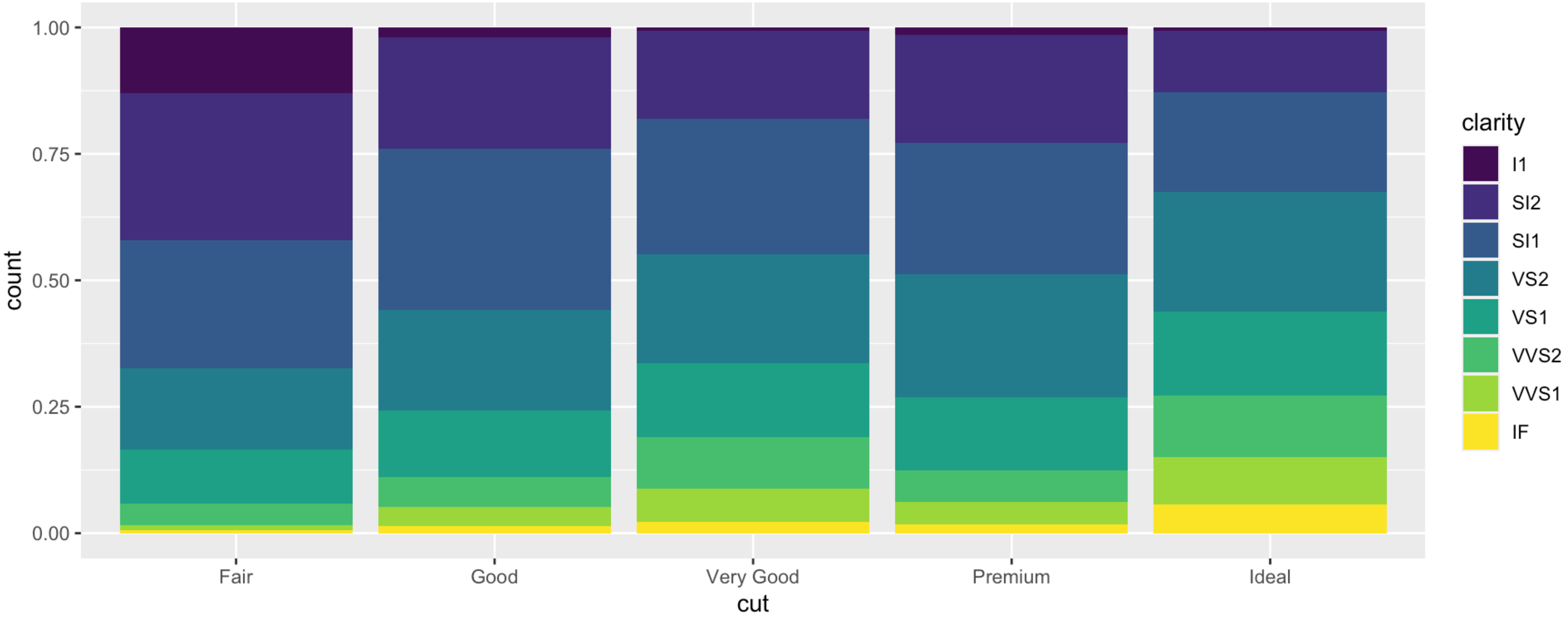
The stacking is performed automatically by the **position adjustment** specified by the `position` argument. If you don’t want a stacked bar chart, you can use one of three other options: `"identity"`, `"dodge"` or `"fill"`.

Position = ‘fill’

`position = "fill"` works like stacking, but makes each set of stacked bars the same height.

This makes it easier to compare proportions across groups.

```r
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity), position = "fill")
```

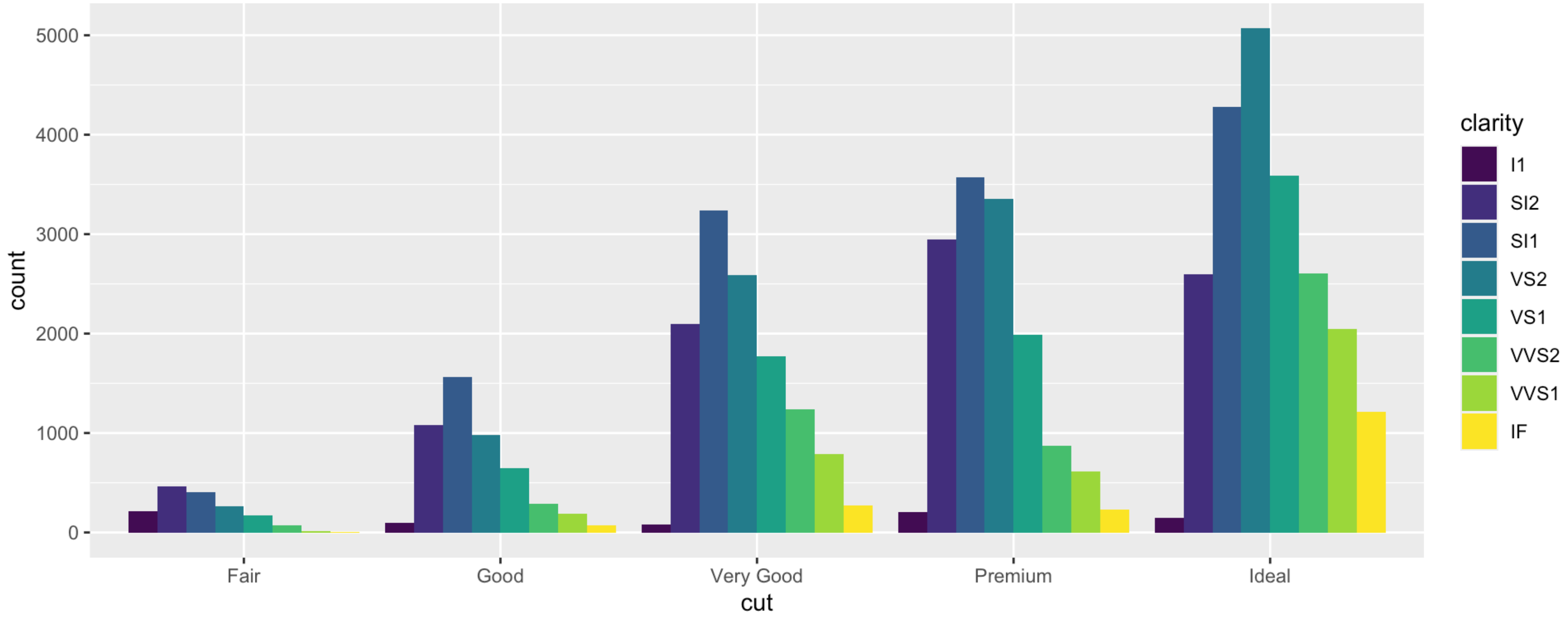


Position = ‘dodge’

`position = "dodge"` places overlapping objects directly *beside* one another.

This makes it easier to compare individual values.

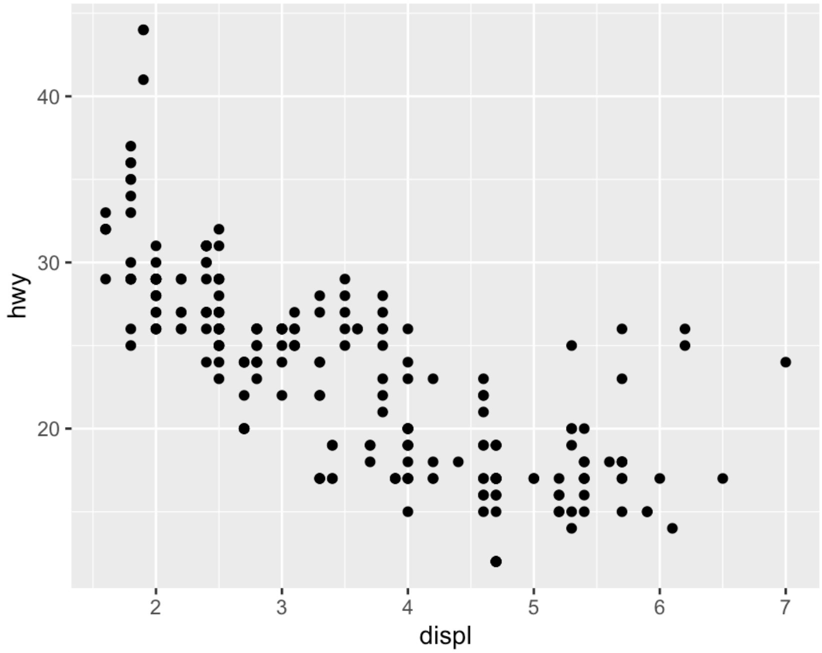
```r
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity), position = "dodge")
```

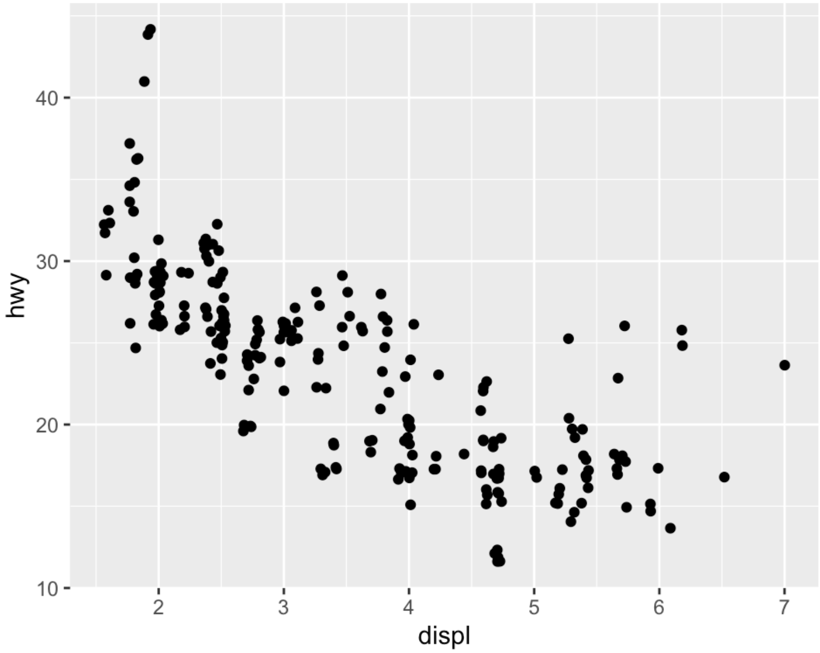


Position = ‘jitter’

```r
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ,
                           y = hwy))
```

```r
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ,
                           y = hwy),
             position = "jitter")
```





Coordinate systems

Coordinate systems are probably the most complicated part of ggplot2.

The default coordinate system is the Cartesian coordinate system where the x and y positions act independently to determine the location of each point.

There are a number of other coordinate systems that are occasionally helpful.

- `coord_flip()` switches the x and y axes.
- `coord_quickmap()` sets the aspect ratio correctly for maps.
- `coord_polar()` uses polar coordinates.

The layered grammar of graphics

Here is an updated template for ggplot code:

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
     mapping = aes(<MAPPINGS>),
     stat = <STAT>,
     position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION>
```

The new template takes seven parameters, the bracketed words that appear in the template.

In practice, you rarely need to supply all seven parameters to make a graph because ggplot2 will provide useful defaults for everything except the data, the mappings, and the geom function.

The seven parameters in the template compose the grammar of graphics, a formal system for building plots.

The grammar of graphics is based on the insight that you can uniquely describe *any* plot as a combination of a dataset, a geom, a set of mappings, a stat, a position adjustment, a coordinate system, and a faceting scheme.