DATA SCIENCE AND ANALYTICS

Week 1

Ayşe Ceren Çiçek

Data Visualization

Most of the code examples written down below, taken from this website https://r4ds.had.co.nz/data-visualisation.html

TABLE OF CONTENTS

Import the library

You only need to install a package once, but you need to reload it every time you start a new session.

```r
#install.packages("tidyverse") to install package
library(tidyverse)
```

Dataset

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

We will use mgs dataset which contains contains a subset of the fuel economy data and only models which had a new release every year between 1999 and 2008.

```r
mpg
```

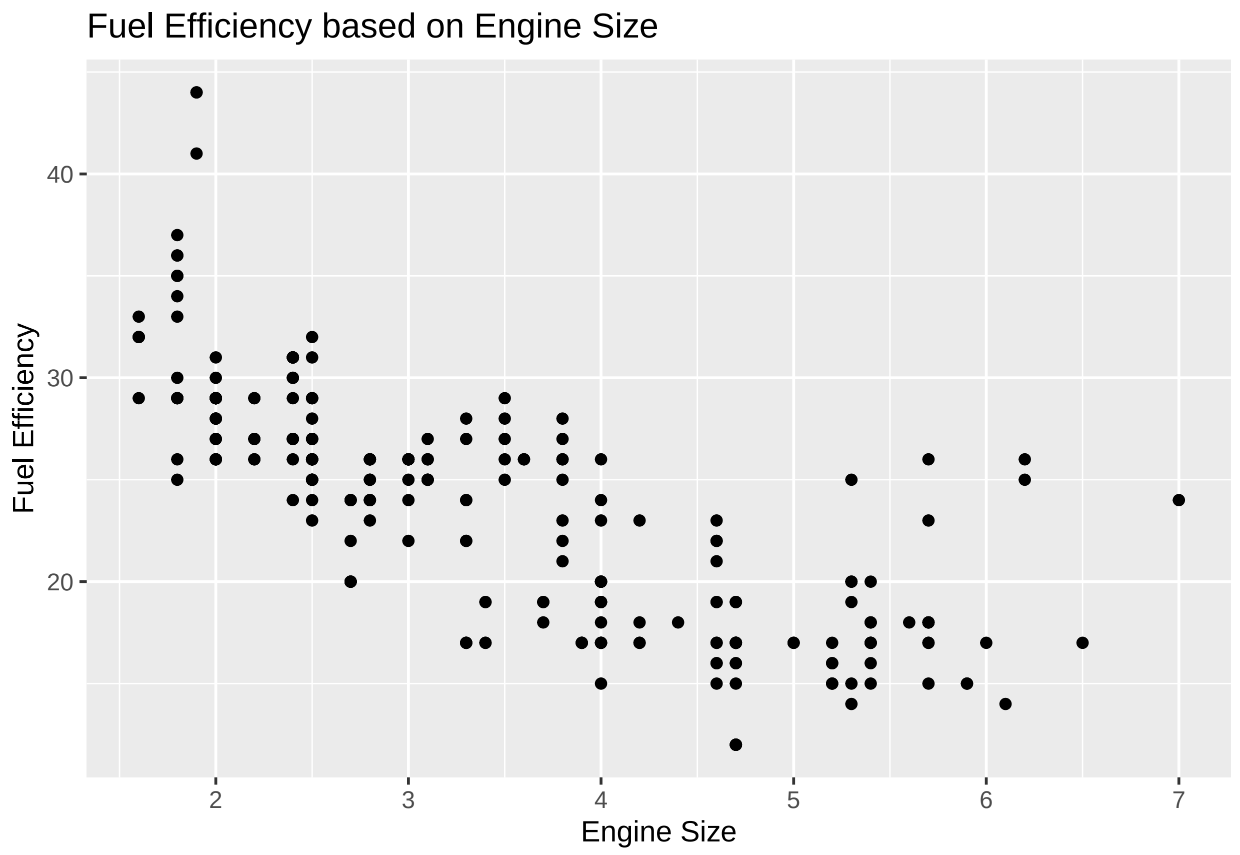
```
## # A tibble: 234 x 11
##    manufacturer model    displ  year   cyl trans   drv     cty   hwy fl    class
##    <chr>        <chr>    <dbl> <int> <int> <chr>   <chr> <int> <int> <chr> <chr>
##  1 audi         a4         1.8  1999     4 auto(l~ f        18    29 p     comp~
##  2 audi         a4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

The columns of the data set are:

- manufacturer: manufacturer name
- model: model name
- displ: engine displacement, in liters
- year: year of manufacture
- cyl: number of cylinders
- trans: type of transmission
- drv: the type of drive train, where f = front-wheel drive, r = rear wheel drive, 4 = 4wd
- cty: city miles per gallon
- hwy: highway miles per gallon (a car's fuel efficiency on the highway)
- fl: fuel type
- class: "type" of car

Plotting with ggplot

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  labs(title = "Fuel Efficiency based on Engine Size",x= "Engine Size",y= "Fuel Efficiency")
```

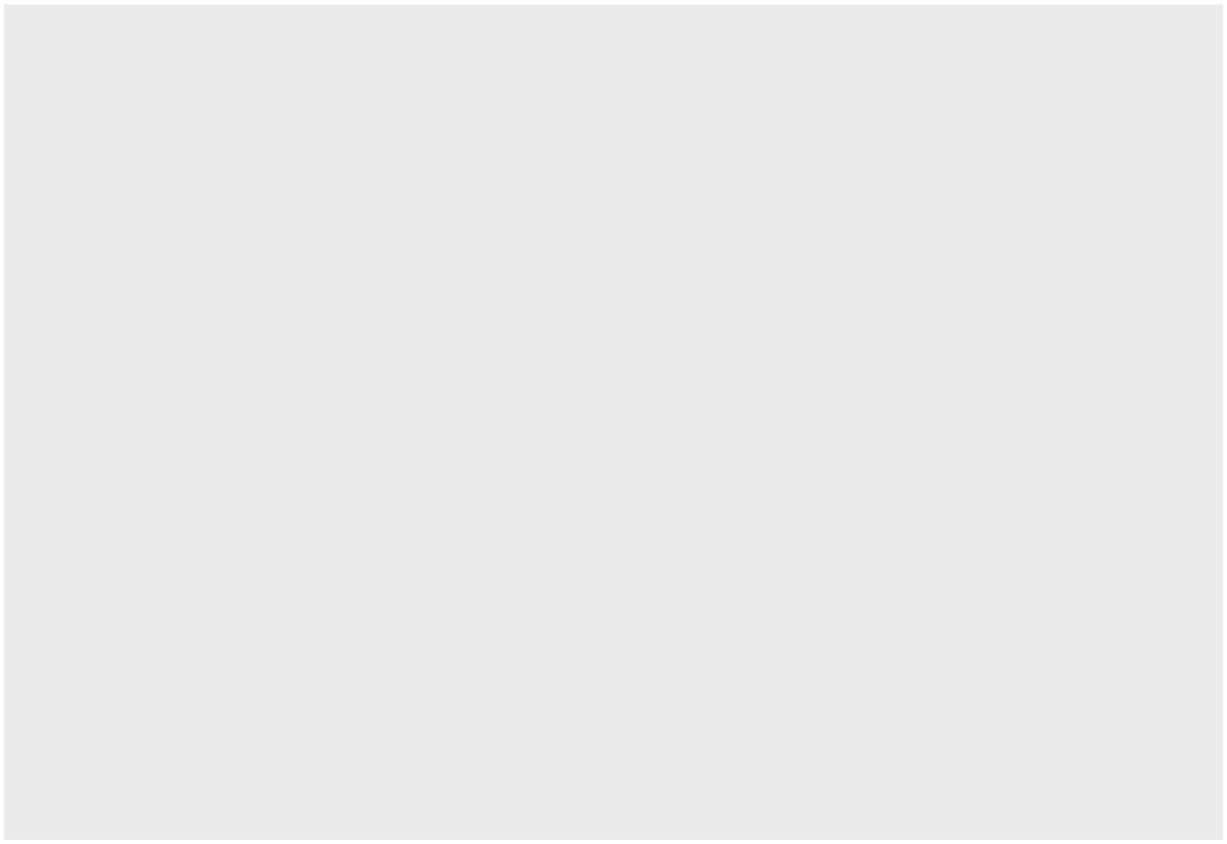


We can see that there is a negative relationship between engine size and fuel efficiency. Which means cars with big engines consumes more fuel.

Exercises

1) Run ggplot(data = mpg). What do you see?

```
ggplot(data = mpg)
```



It shows an empty plot.

2) How many rows are in mtcars? How many columns?

```
nrow(mtcars)
```

```
## [1] 32
```

```
ncol(mtcars)
```

```
## [1] 11
```

mtcars dataset contains 32 rows and 11 columns.

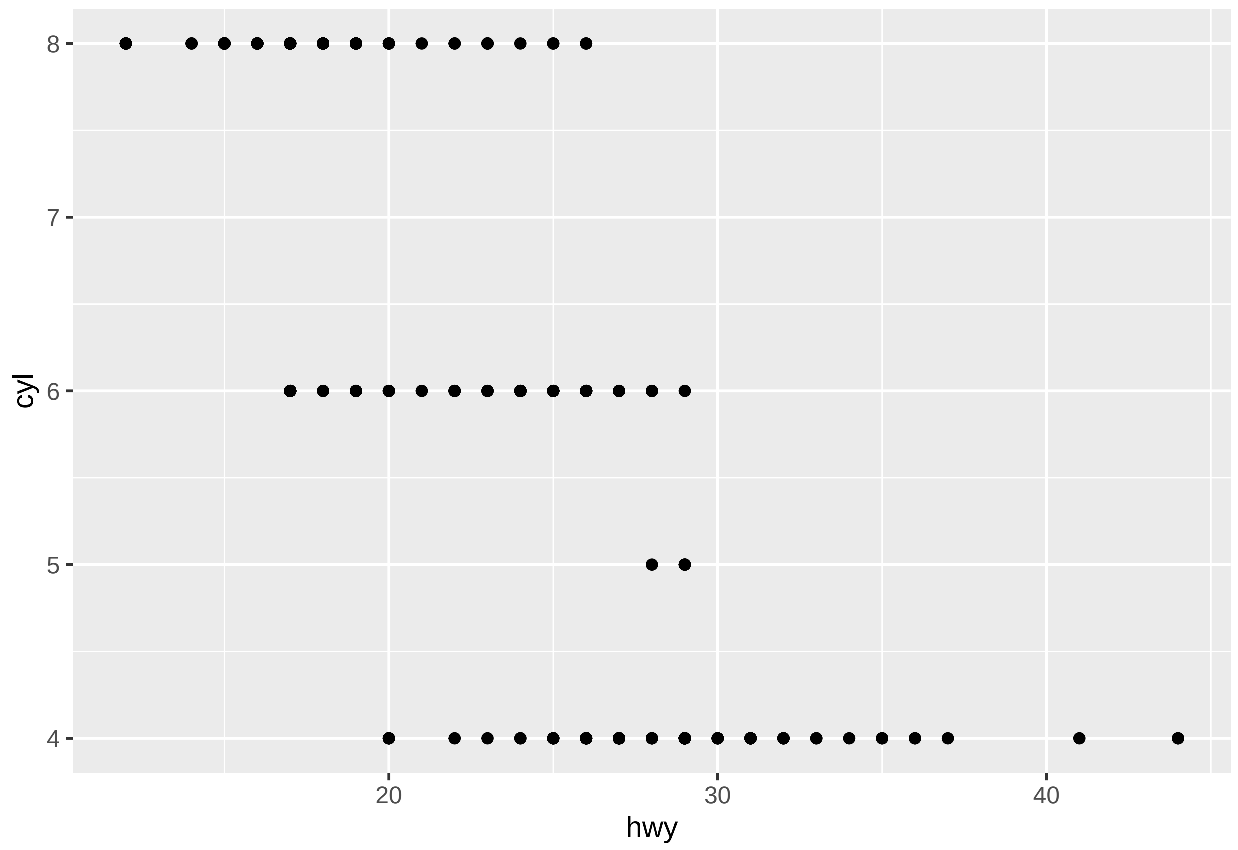
3) What does the drv variable describe? Read the help for ?mpg to find out.

```
help(mpg)
```

drv variable describes the type of drive train, where f = front-wheel drive, r = rear wheel drive, 4 = 4wd.

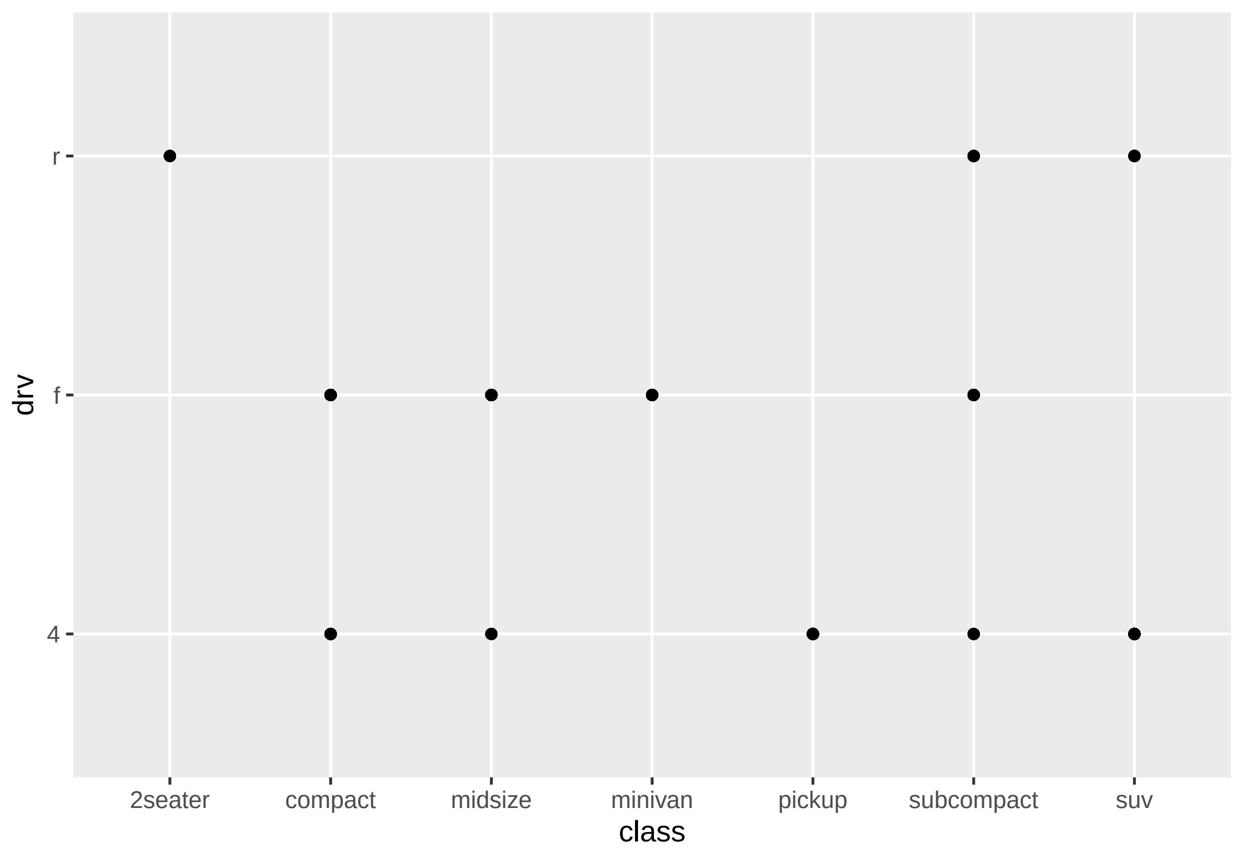
4) Make a scatterplot of hwy versus cyl.

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



5) What happens if you make a scatterplot of class versus drv? Why is the plot not useful?

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = class, y = drv))
```

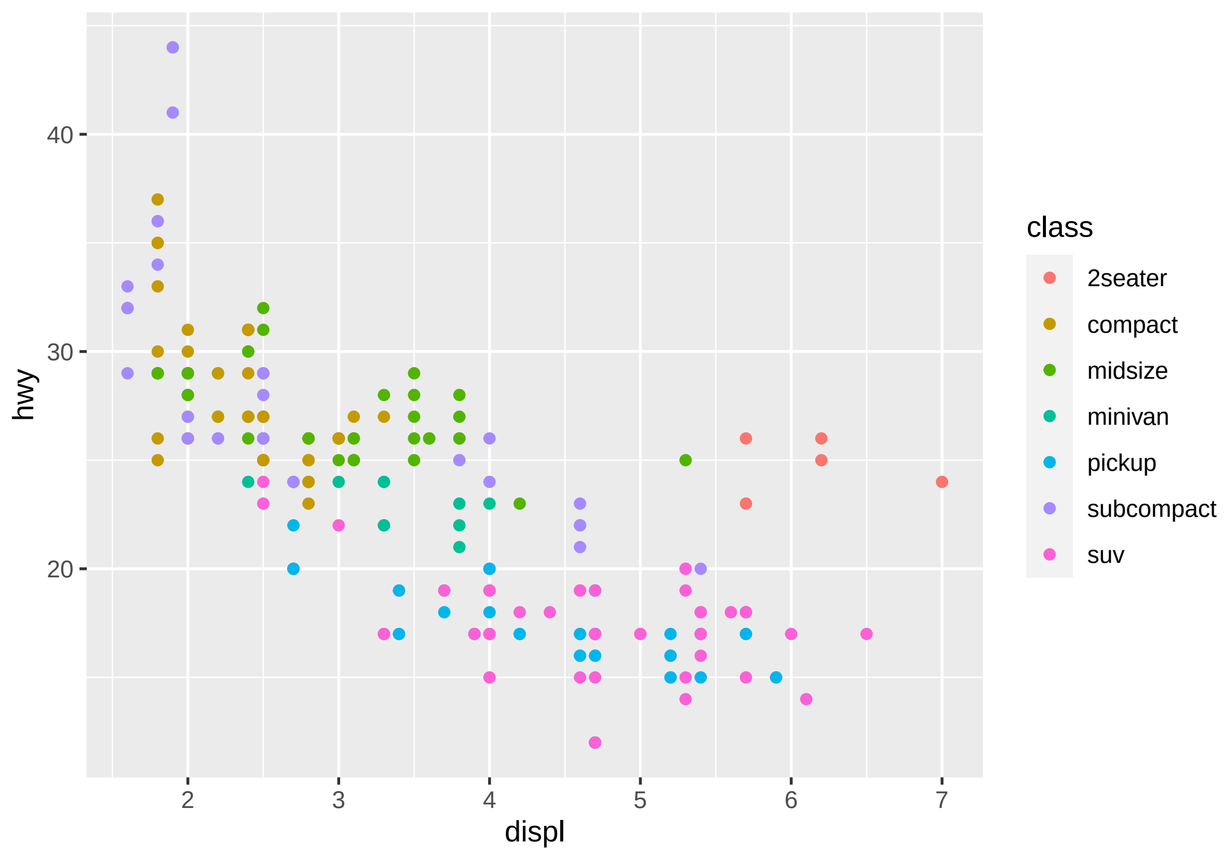


We can not define a relationship.

Aesthetic Mappings

Aesthetic mappings describe how variables in the data are mapped to visual properties (aesthetics) of geoms. We can change point's size, shape and color.

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

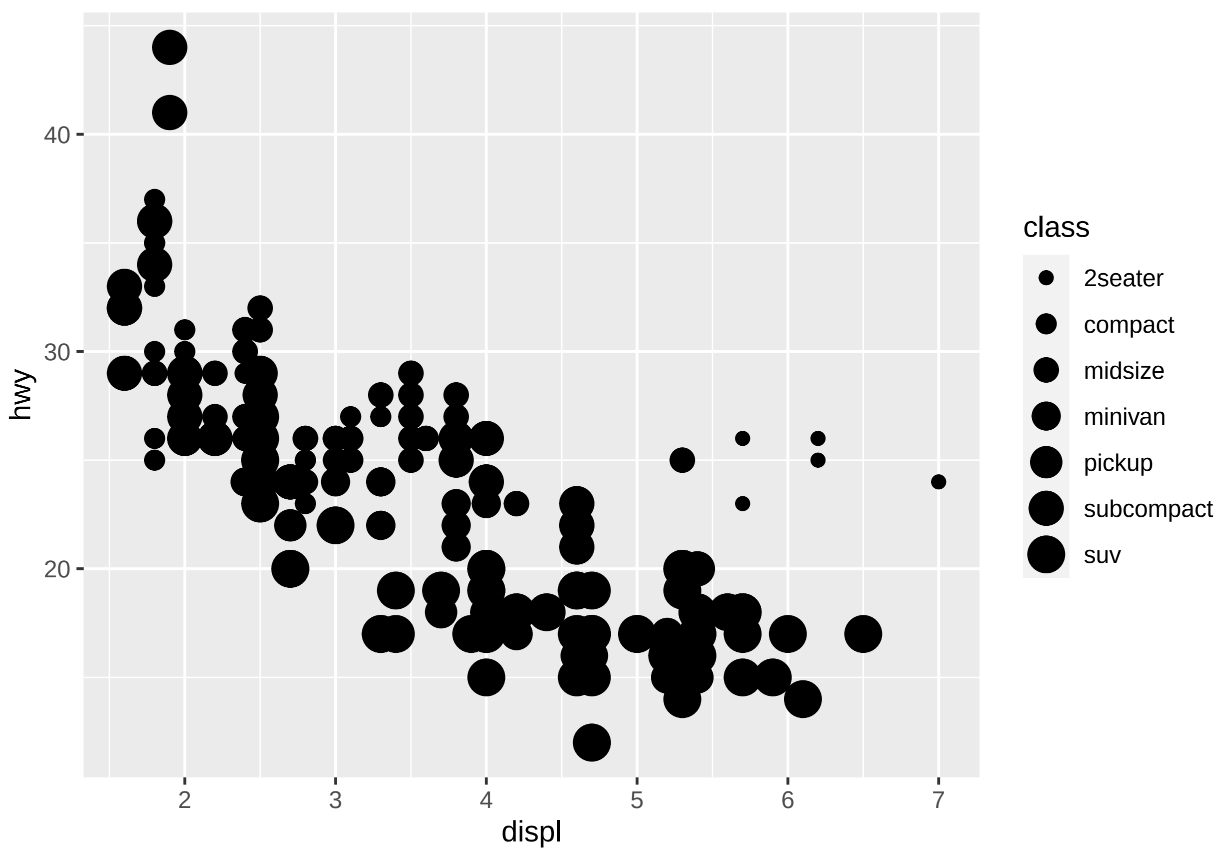


This plot maps colors of the points to classes. we can see which color corresponds to which class on the legend. For example, pink points represents the class suv.

We can map classes to size aesthetic. But mapping an unordered variable (class) to an ordered aesthetic (size) is not recommended.

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

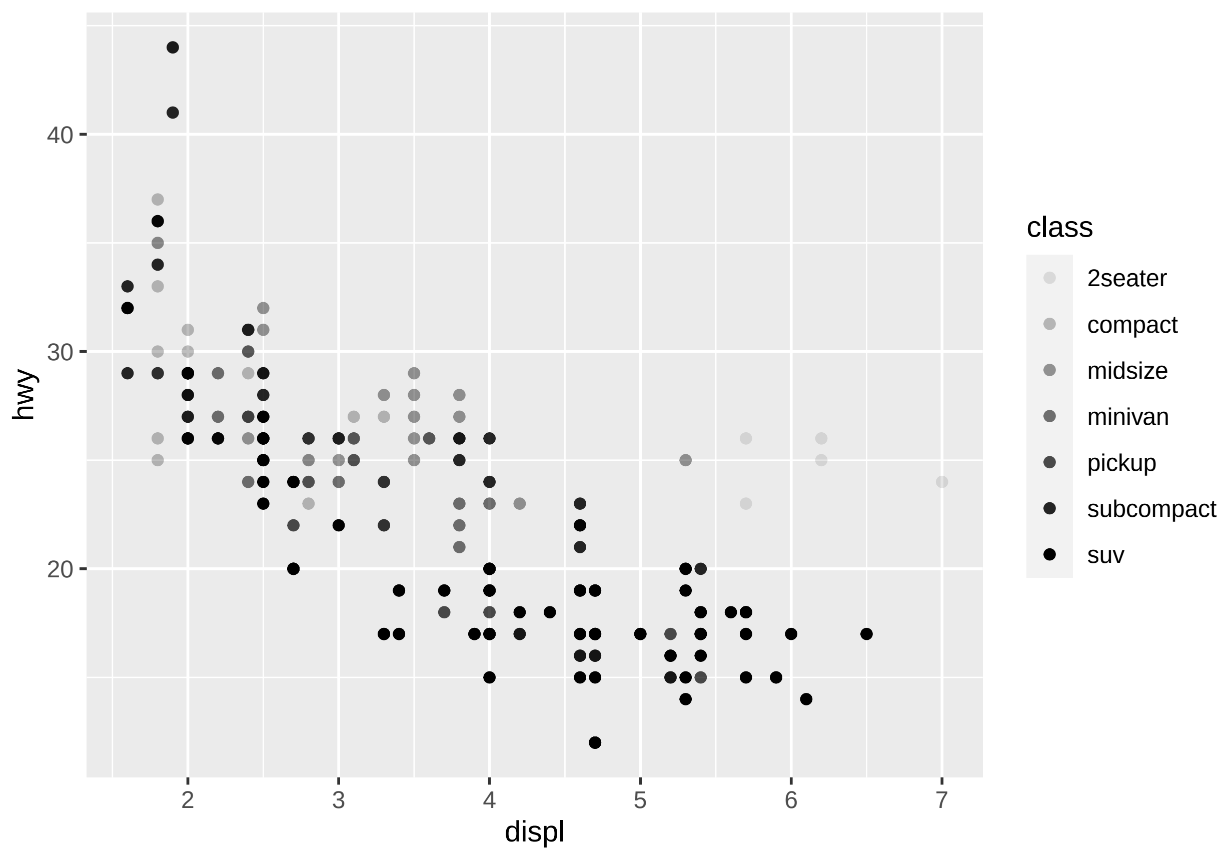
```
## Warning: Using size for a discrete variable is not advised.
```



We can also map class to the alpha aesthetic, which helps us to change the transparency or the shape of the points.

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

```
## Warning: Using alpha for a discrete variable is not advised.
```

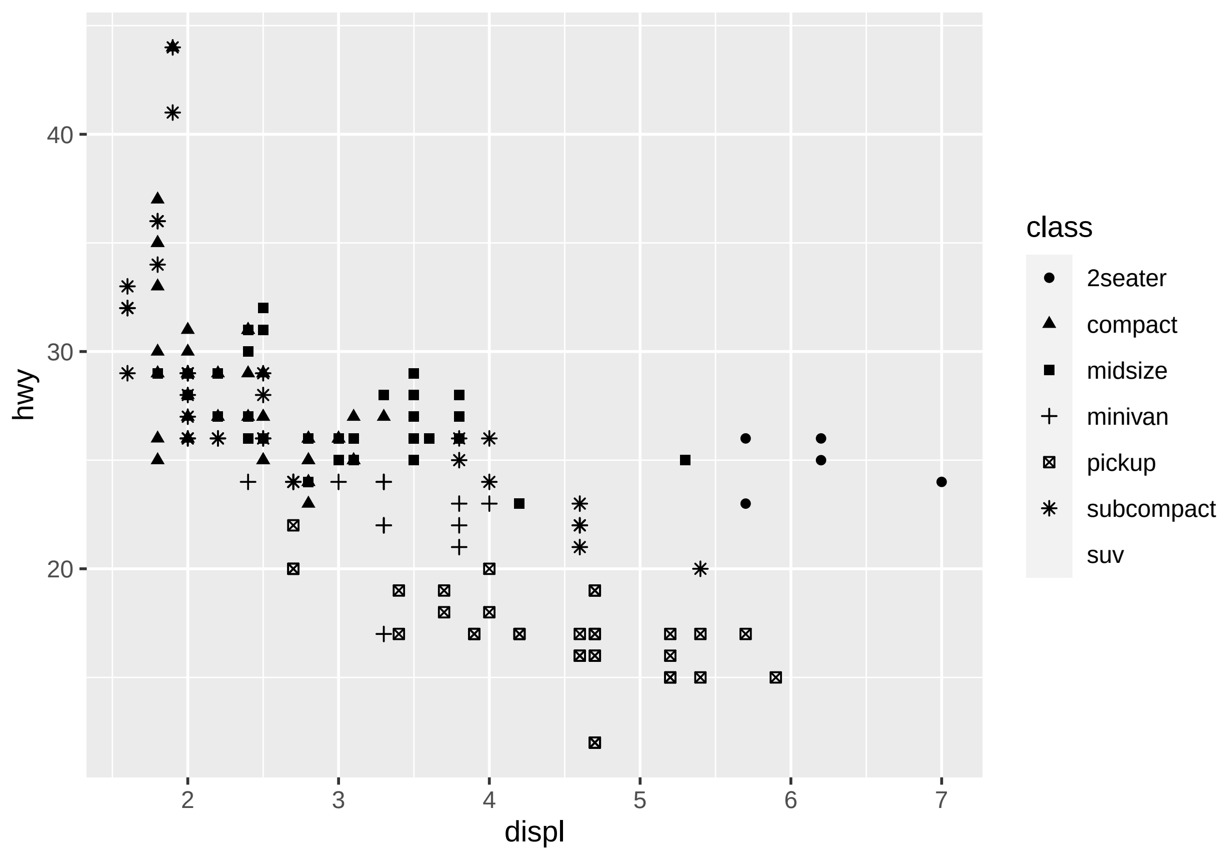


The shape can group maximum 6 classes, so the additional groups like suv class will not be plotted.

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

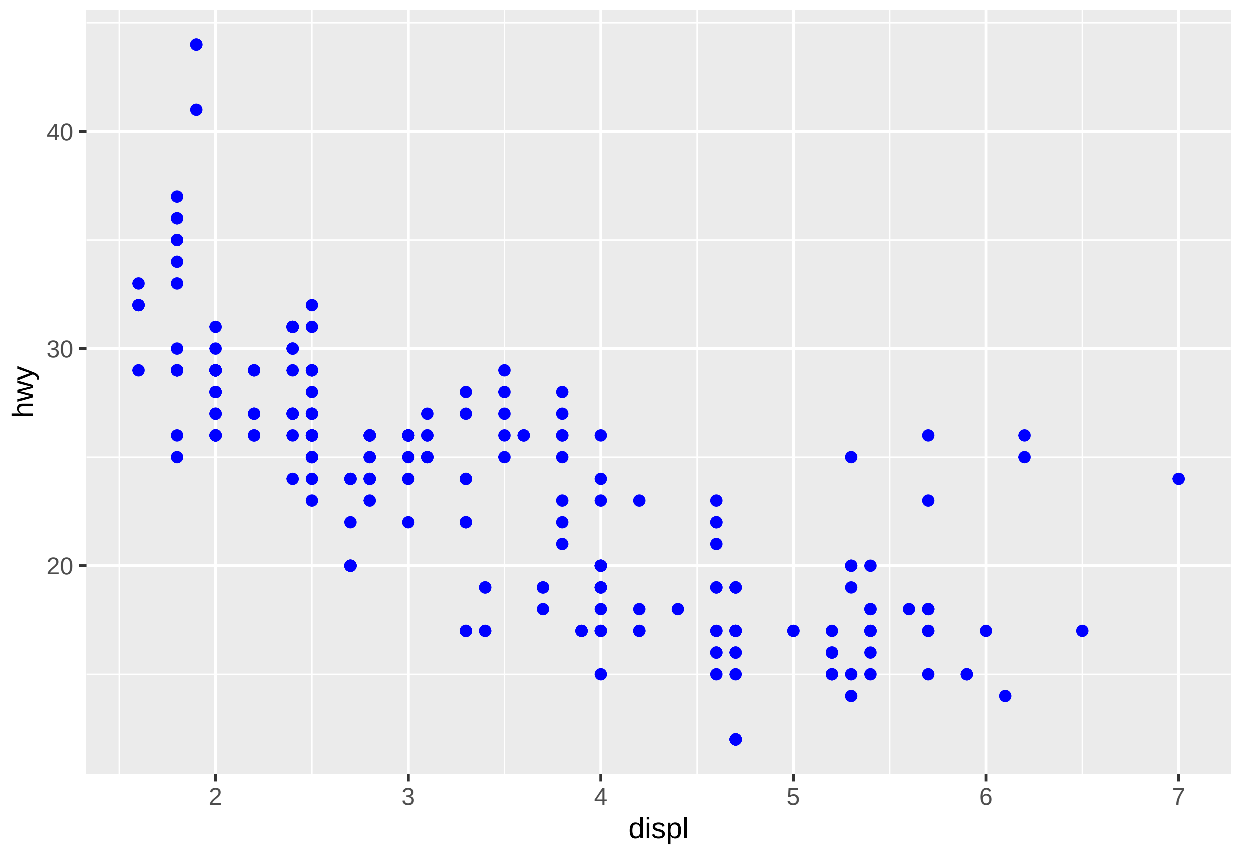
```
## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 7. Consider
## specifying shapes manually if you must have them.
```

```
## Warning: Removed 62 rows containing missing values (geom_point).
```



With color parameter, we can change the colors of the points.

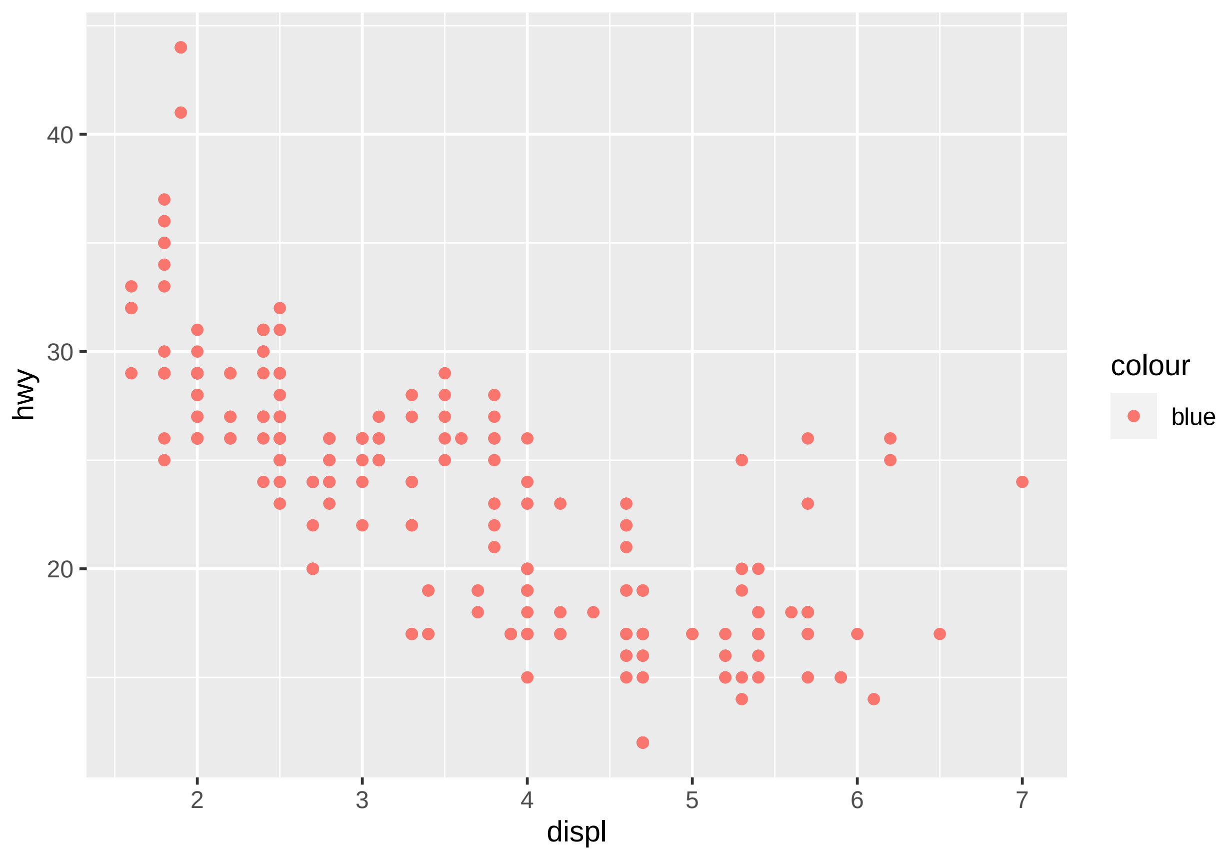
```r
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), color = "blue")
```



Exercises 2

1) What's gone wrong with this code? Why are the points not blue?

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



The color does not convey information about a variable, it changes the appearance of the plot. We should write color outside of the aes function.

3) Which variables in mpg are categorical? Which variables are continuous?

(Hint: type ?mpg to read the documentation for the dataset.) How can you see this information when you run mpg?

```
summary(mpg)
```

```
##  manufacturer          model               displ            year     
##  Length:234         Length:234         Min.   :1.600   Min.   :1999  
##  Class :character   Class :character   1st Qu.:2.400   1st Qu.:1999  
##  Mode  :character   Mode  :character   Median :3.300   Median :2004  
##                                        Mean   :3.472   Mean   :2004  
##                                        3rd Qu.:4.600   3rd Qu.:2008  
##                                        Max.   :7.000   Max.   :2008  
##       cyl           trans                drv                 cty       
##  Min.   :4.000   Length:234         Length:234         Min.   : 9.00  
##  1st Qu.:4.000   Class :character   Class :character   1st Qu.:14.00  
##  Median :6.000   Mode  :character   Mode  :character   Median :17.00  
##  Mean   :5.889                                         Mean   :16.86  
##  3rd Qu.:8.000                                         3rd Qu.:19.00  
##  Max.   :8.000                                         Max.   :35.00  
##       hwy             fl               class          
##  Min.   :12.00   Length:234         Length:234        
##  1st Qu.:18.00   Class :character   Class :character
##  Median :24.00   Mode  :character   Mode  :character
##  Mean   :23.44
##  3rd Qu.:27.00
##  Max.   :44.00
```

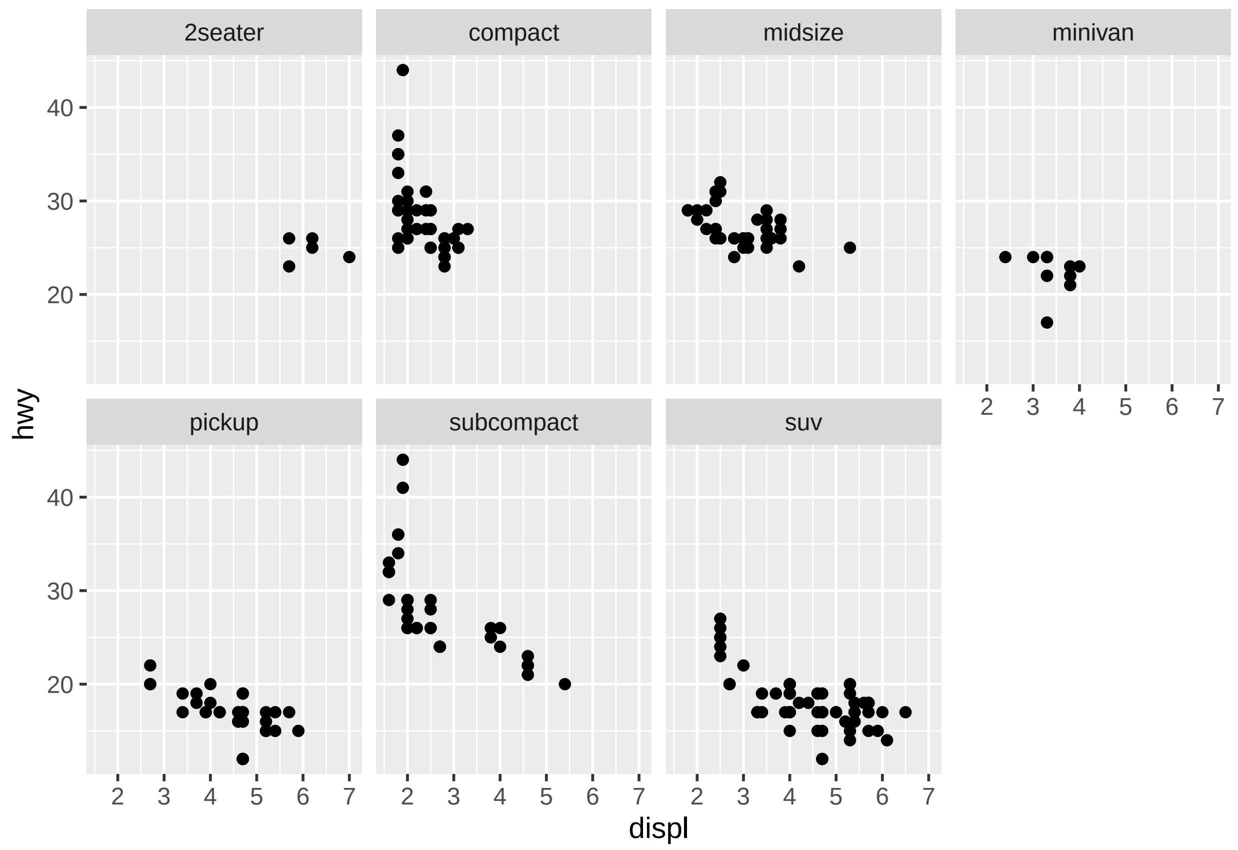
6 variables are categorical which are manufacturer, model, trans, drv, fl, class.

Facets

For categorical variables, we can split our plot into facets, which are subplots that each display one subset of the data.

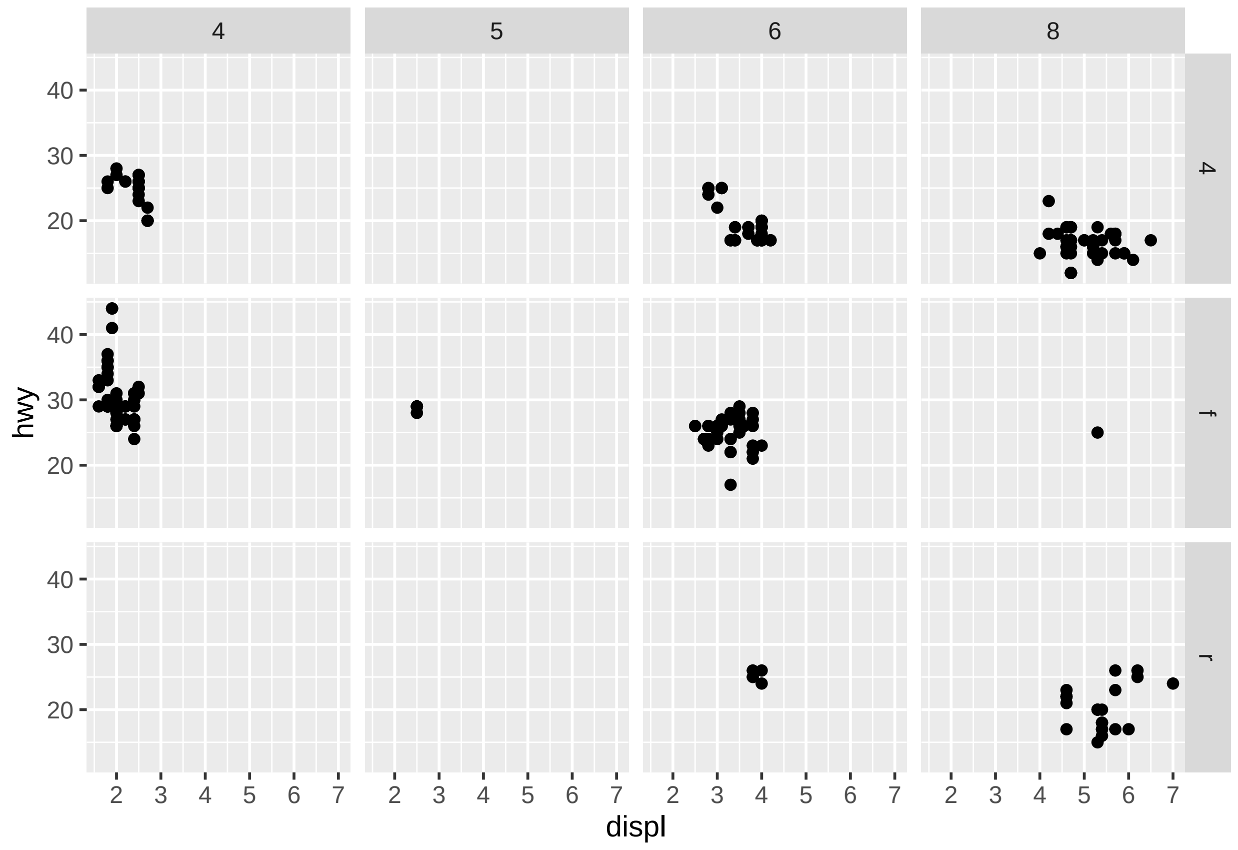
facet_wrap() function facets the plot by a single variable.

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



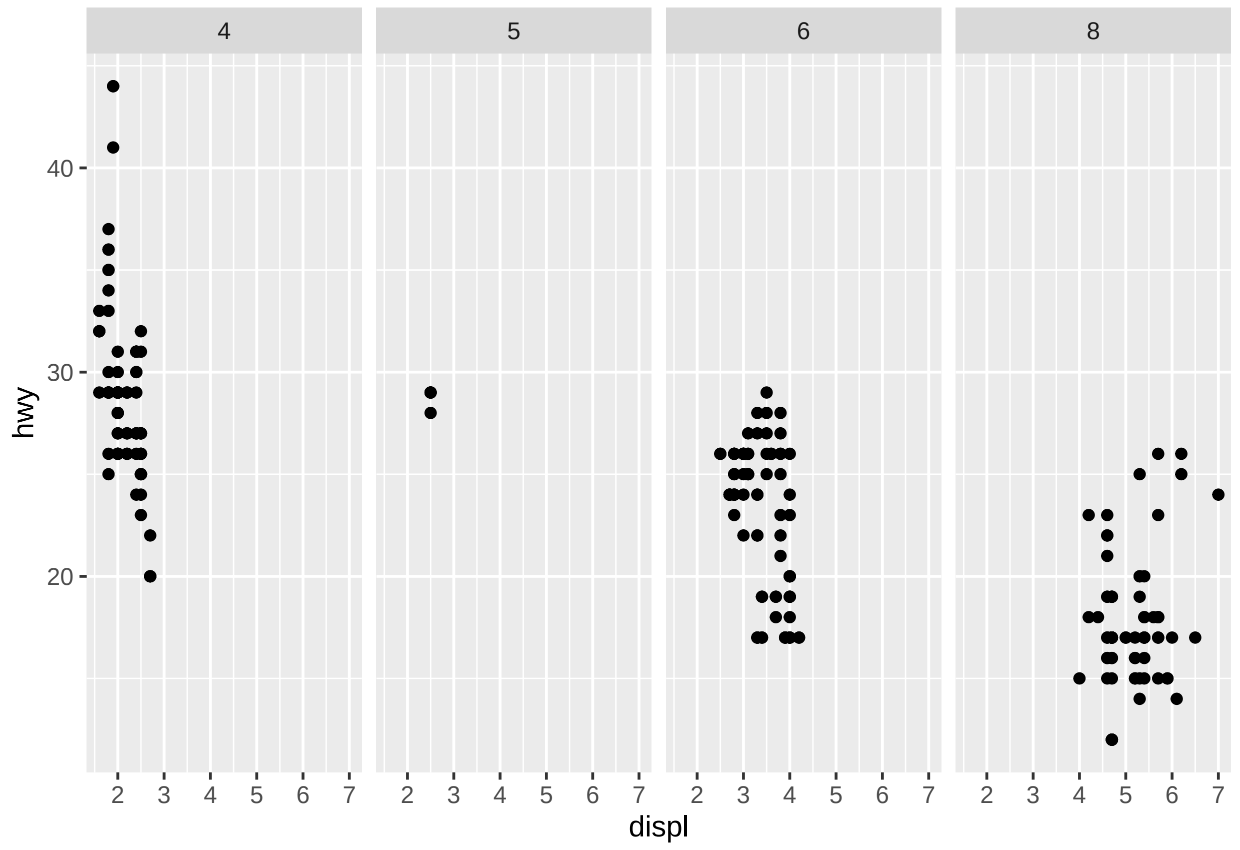
facet_grid() facets your plot on the combination of two variables.

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



To not facet in the rows or columns dimension, use a . instead of a variable name.

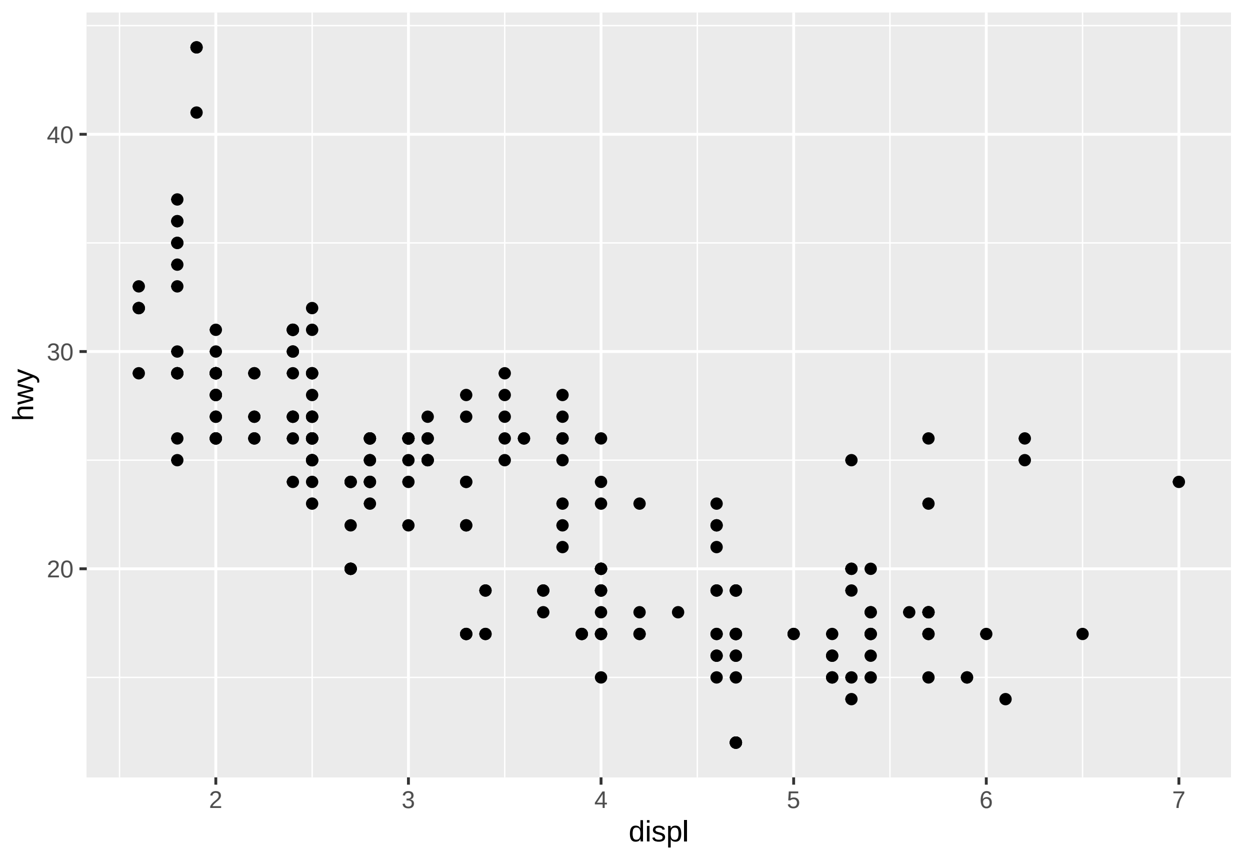
```r
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_grid(. ~ cyl)
```



Geometric Objects

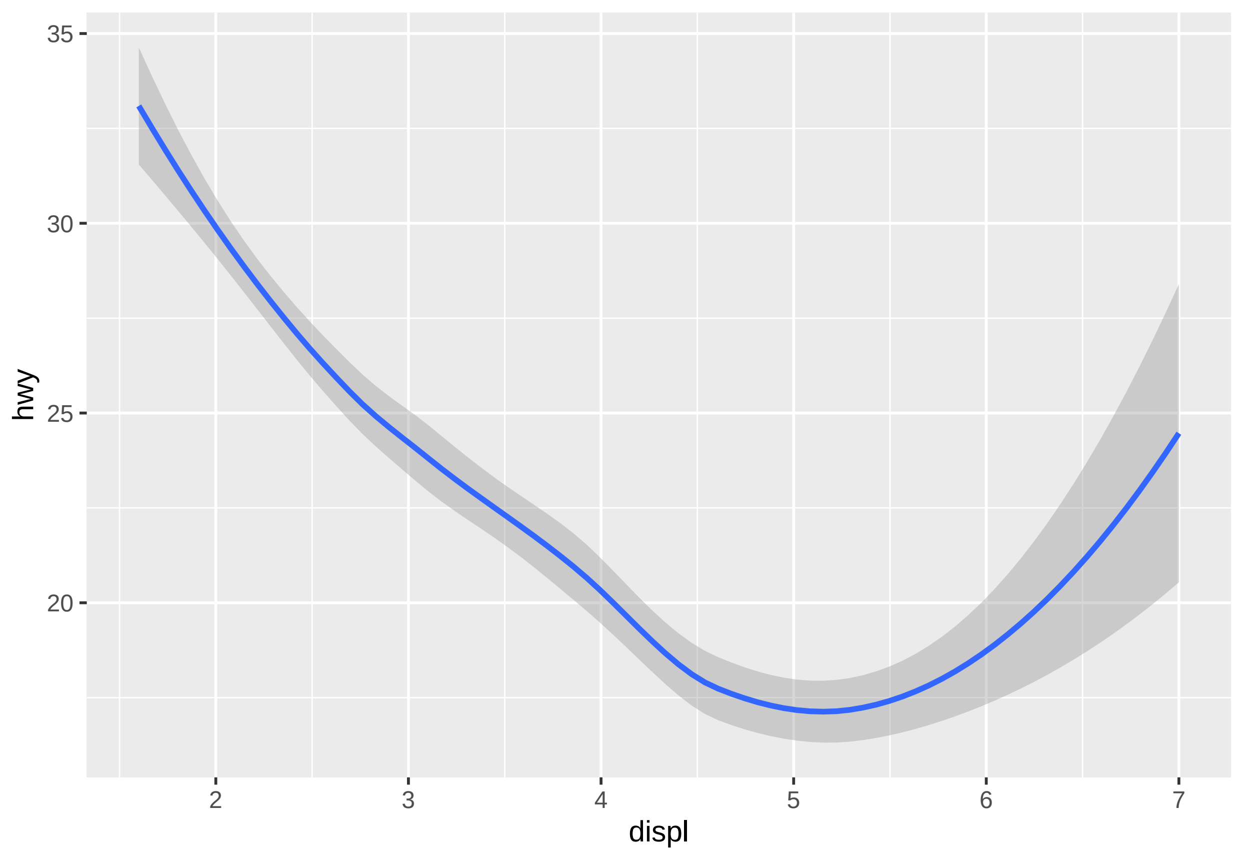
A geom is the geometrical object that a plot uses to represent data. To change the geom in your plot, change the geom function that you add to ggplot().

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



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

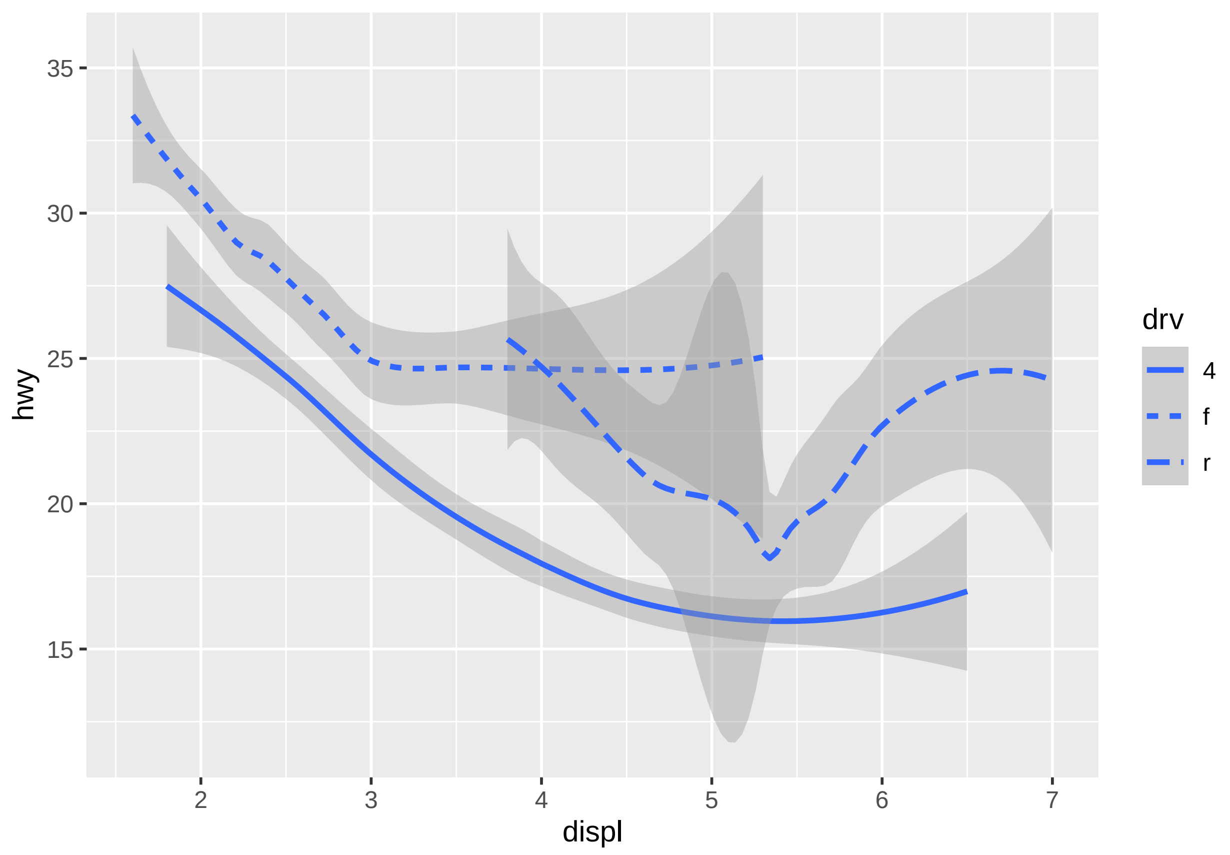
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



The below plot shows that geom_smooth() separates the cars into three lines based on their drv value, which describes a car's drivetrain.

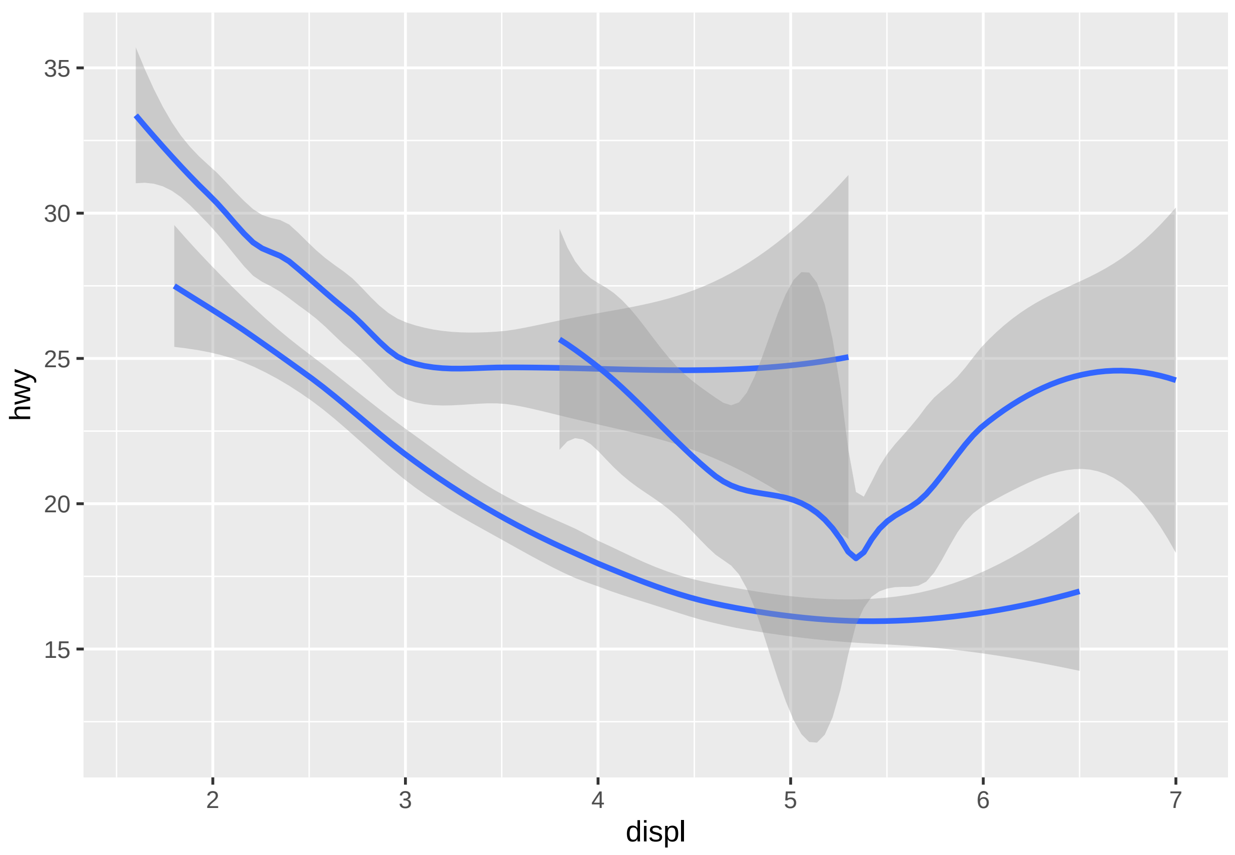
```
ggplot(data = mpg) +
  geom_smooth(mapping = (aes(x = displ, y = hwy, linetype = drv)))
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



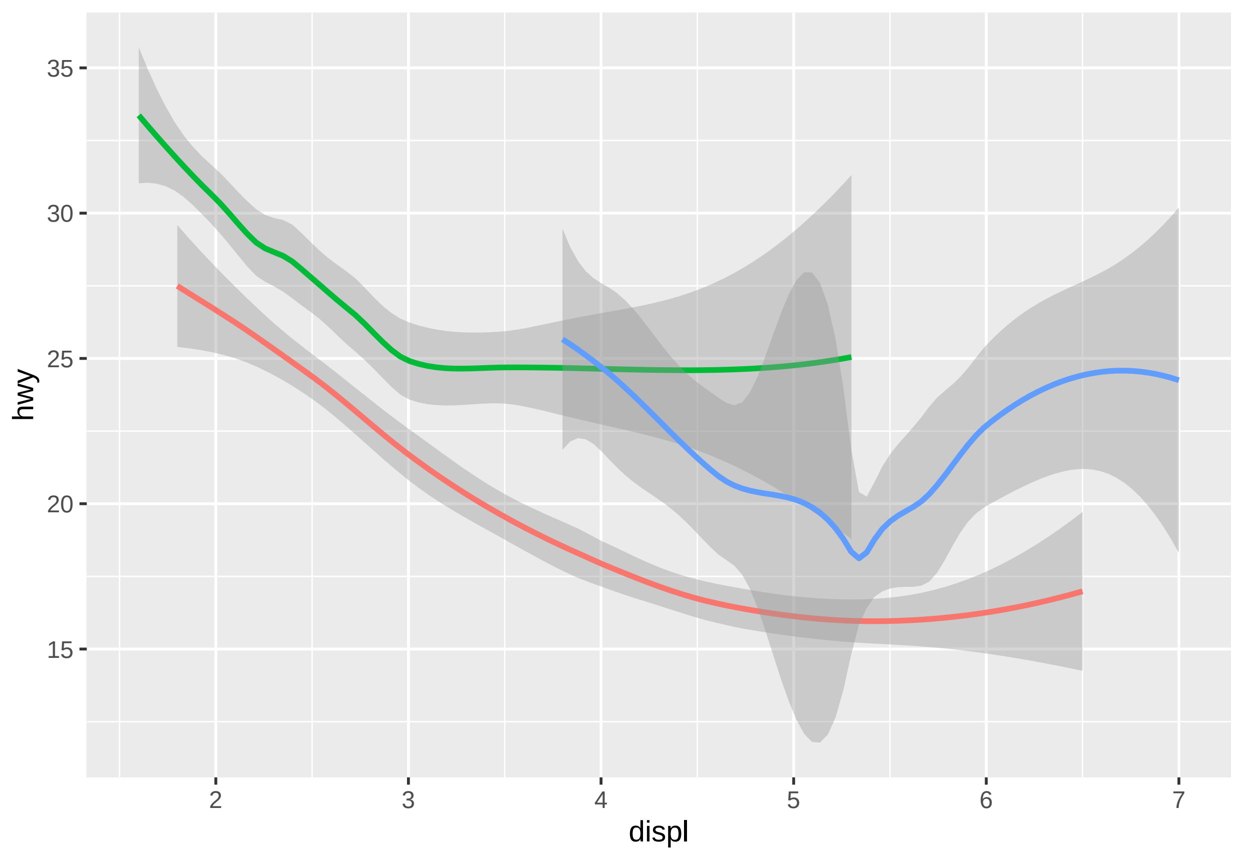
```
ggplot(data = mpg) +
geom_smooth(mapping = aes(x = displ, y = hwy, group = drv))
```

```
## ‘geom_smooth()‘ using method = ’loess’ and formula ’y ~ x’
```



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

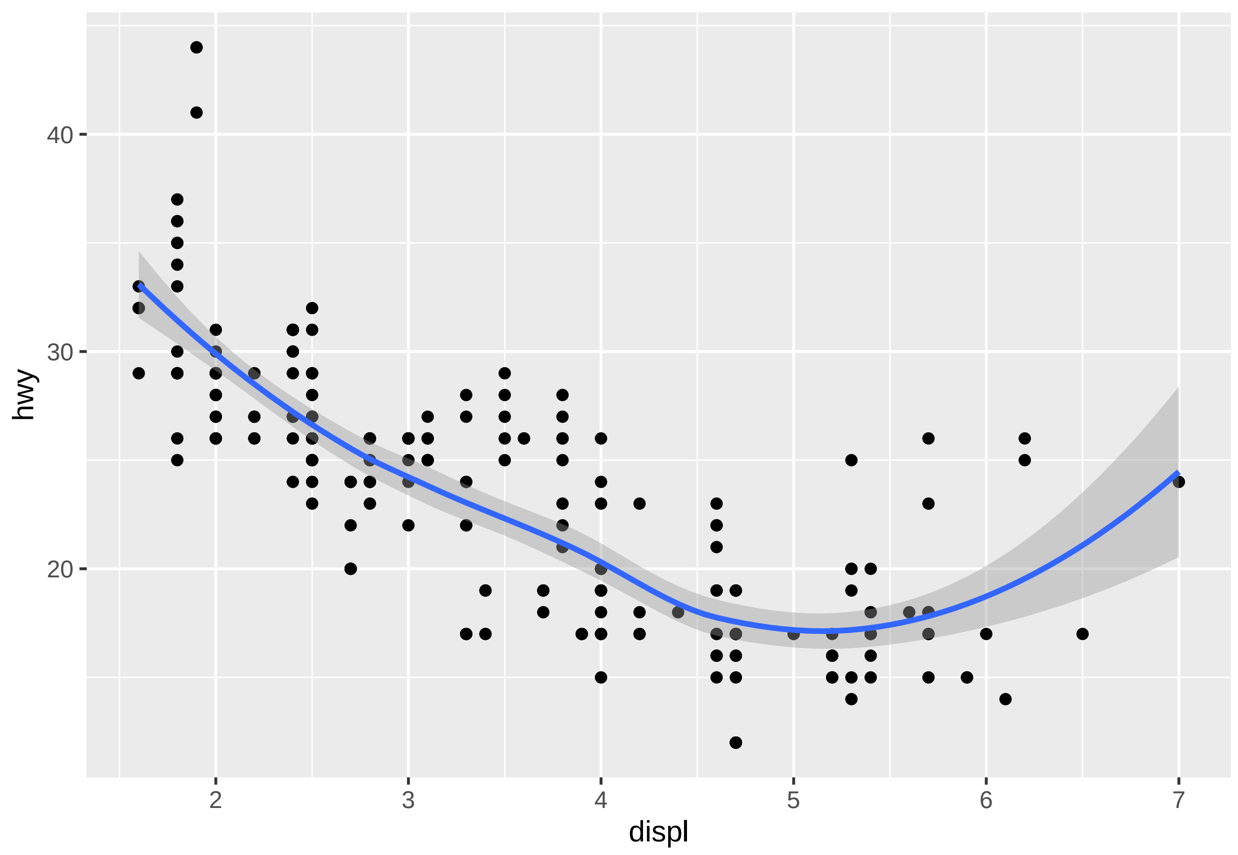
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



To display multiple geoms in the same plot, we should add multiple geom functions.

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

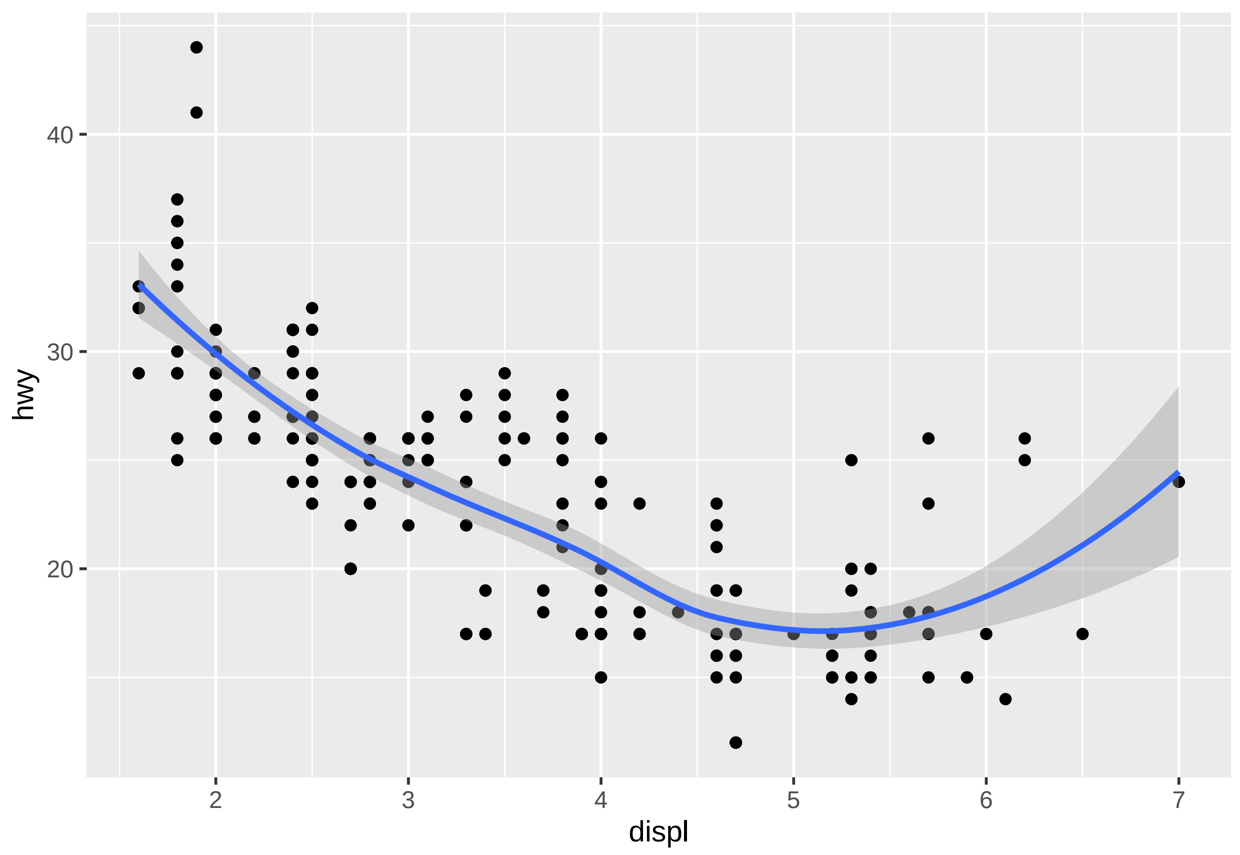
```
## ‘geom_smooth()‘ using method = ’loess’ and formula ’y ~ x’
```



But the code above, includes some duplication. When you want to change y-axis you should change it in both of the geom_functions. To avoid this we can pass set of mappings to ggplot(). The output plot will be same with the previous one.

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

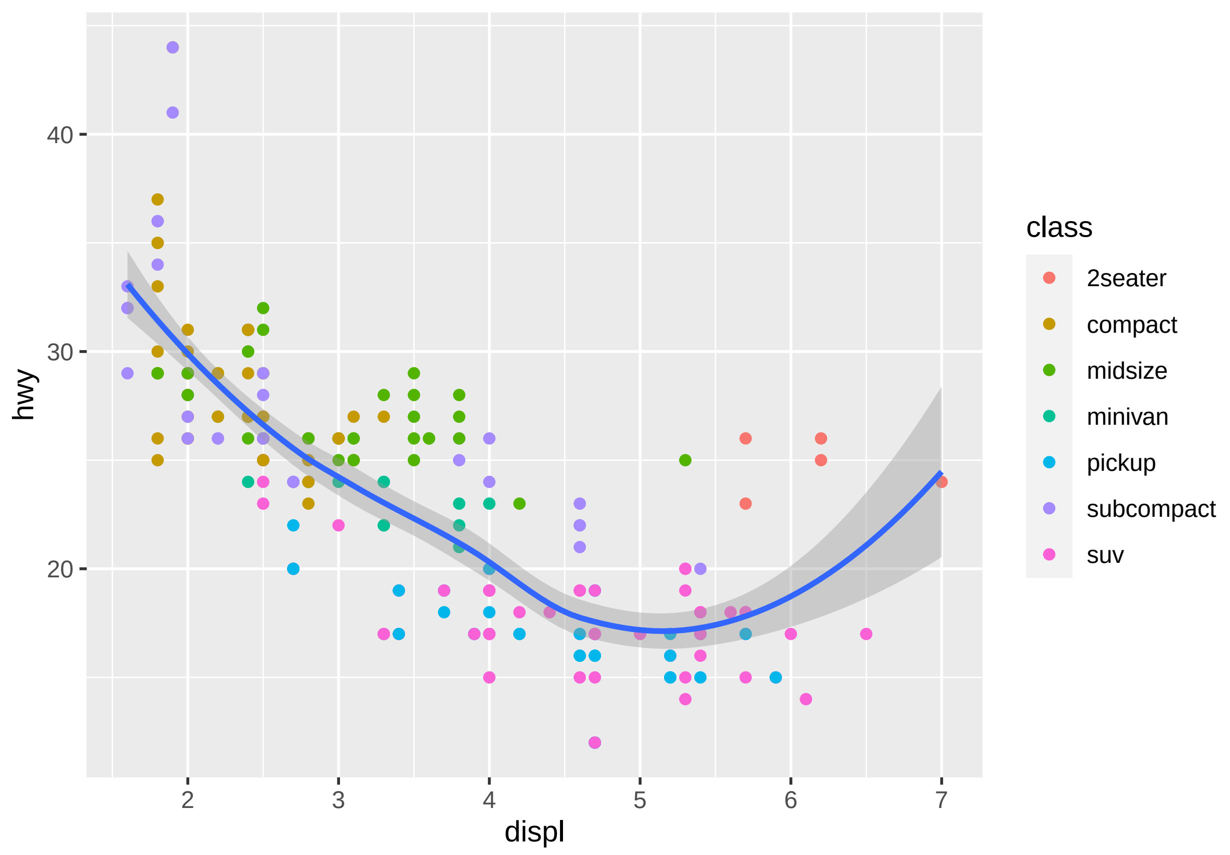
```
## ‘geom_smooth()‘ using method = ’loess’ and formula ’y ~ x’
```



If we add mappings to geom functions, ggplot2 will use these mappings to extend or overwrite the global mappings.

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

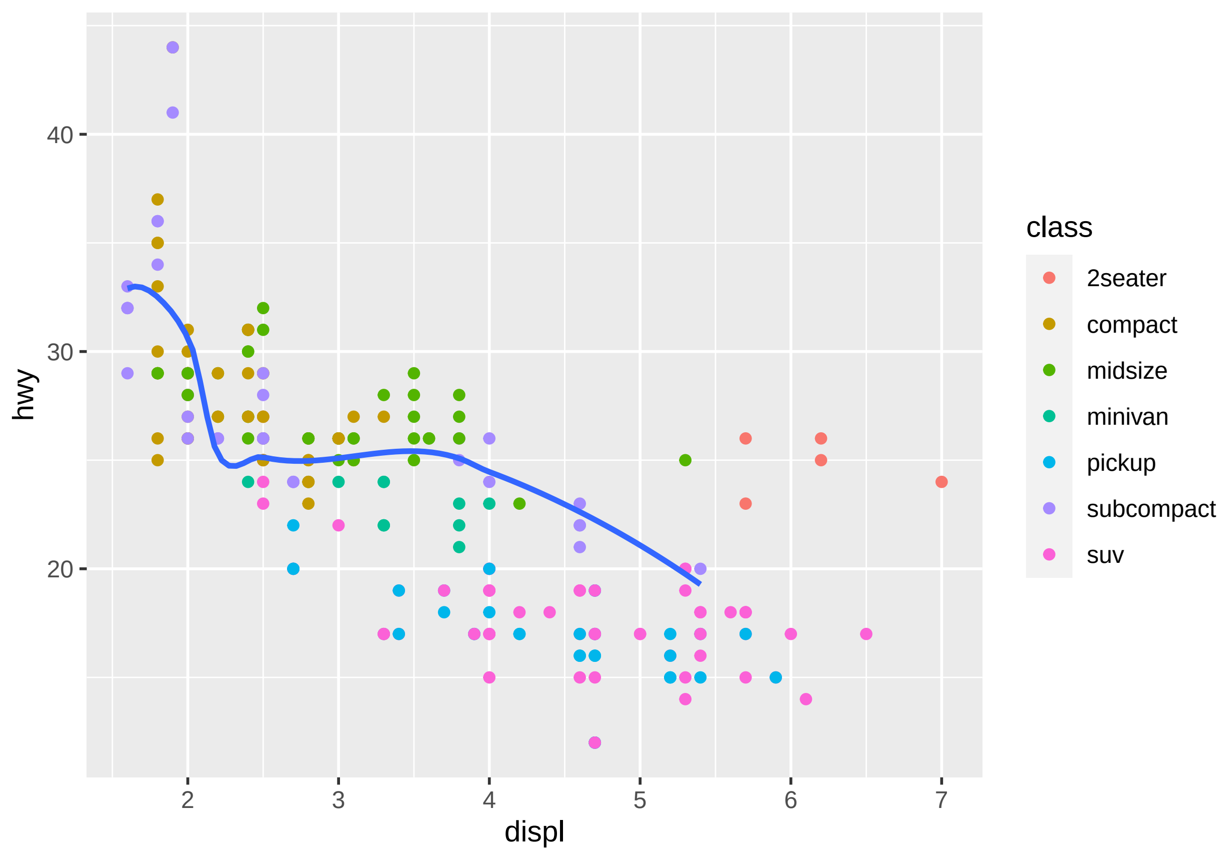
```
## ‘geom_smooth()‘ using method = ’loess’ and formula ’y ~ x’
```



You can use same thing to specify different data for each layer.

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

```
## ‘geom_smooth()‘ using method = ’loess’ and formula ’y ~ x’
```

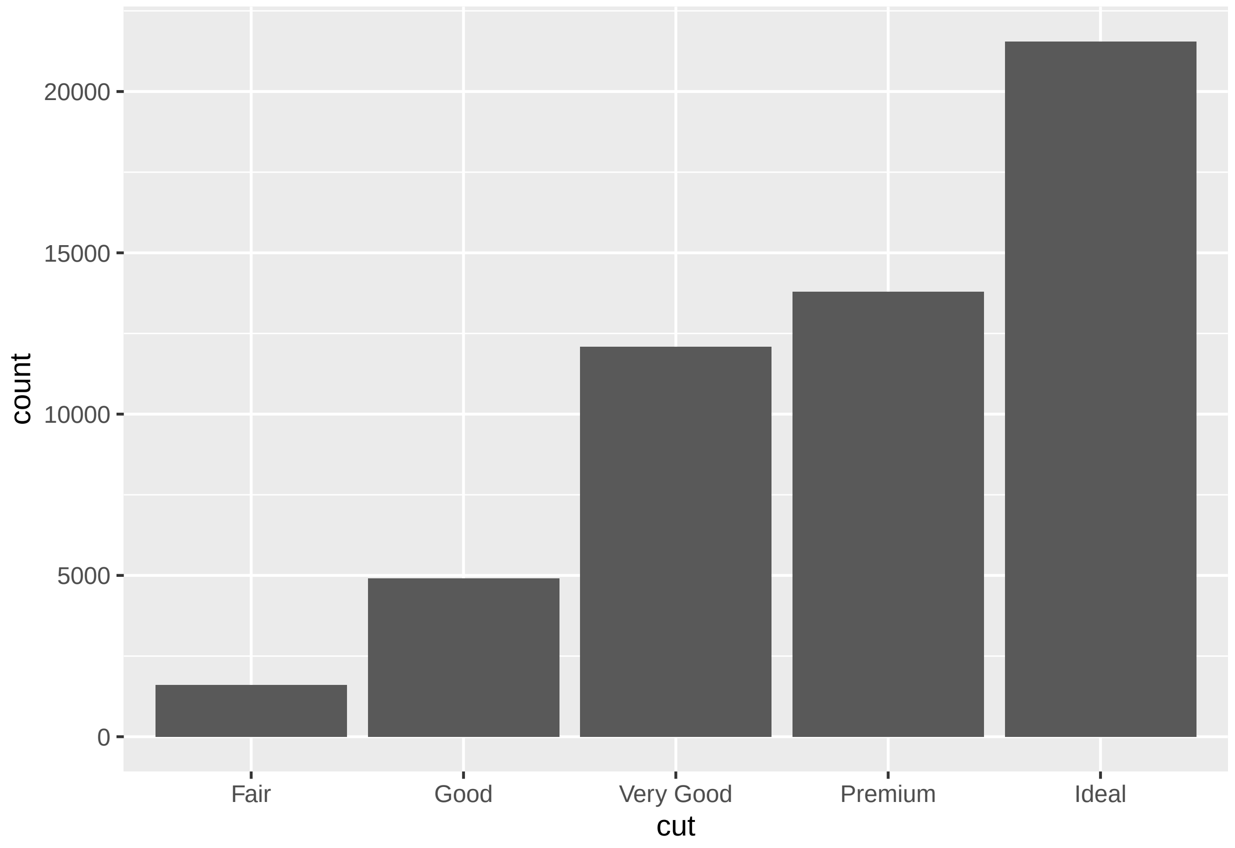


Statistical Transformations

```
summary(diamonds)
```

```
##      carat               cut        color      clarity         depth      
##  Min.   :0.2000   Fair     : 1610   D: 6775   SI1    :13065   Min.   :43.00  
##  1st Qu.:0.4000   Good     : 4906   E: 9797   VS2    :12258   1st Qu.:61.00  
##  Median :0.7000   Very Good:12082   F: 9542   SI2    : 9194   Median :61.80  
##  Mean   :0.7979   Premium  :13791   G:11292   VS1    : 8171   Mean   :61.75  
##  3rd Qu.:1.0400   Ideal    :21551   H: 8304   VVS2   : 5066   3rd Qu.:62.50  
##  Max.   :5.0100                     I: 5422   VVS1   : 3655   Max.   :79.00  
##                                     J: 2808   (Other): 2531                  
##      table           price             x                y         
##  Min.   :43.00   Min.   :  326   Min.   : 0.000   Min.   : 0.000  
##  1st Qu.:56.00   1st Qu.:  950   1st Qu.: 4.710   1st Qu.: 4.720  
##  Median :57.00   Median : 2401   Median : 5.700   Median : 5.710  
##  Mean   :57.46   Mean   : 3933   Mean   : 5.731   Mean   : 5.735  
##  3rd Qu.:59.00   3rd Qu.: 5324   3rd Qu.: 6.540   3rd Qu.: 6.540  
##  Max.   :95.00   Max.   :18823   Max.   :10.740   Max.   :58.900  
##                                                                   
##        z         
##  Min.   : 0.000  
##  1st Qu.: 2.910  
##  Median : 3.530
##  Mean   : 3.539
##  3rd Qu.: 4.040
##  Max.   :31.800
##
```

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

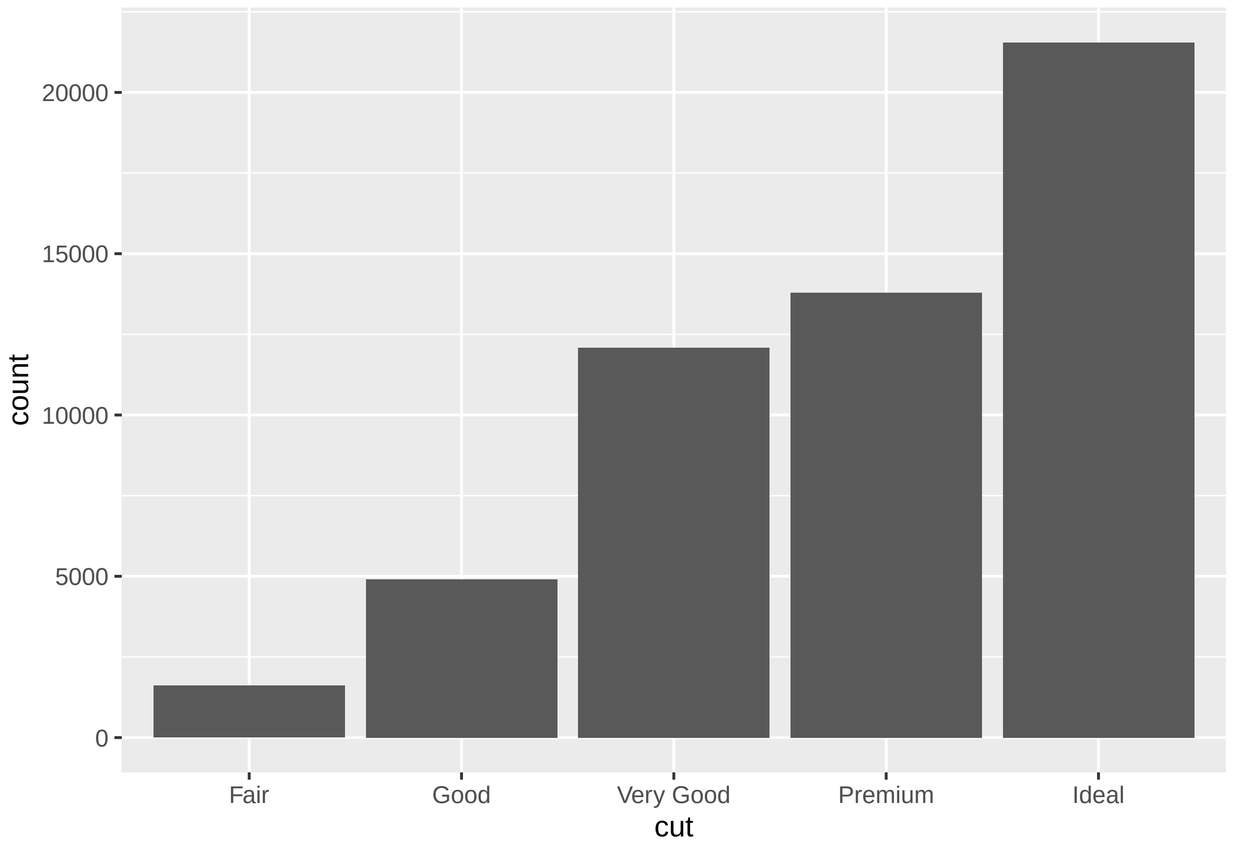


This plot also shows count variable which is not variable in diamonds. Many graphs, like scatterplots, plot the raw values of your dataset. Other graphs, like bar charts, calculate new values to plot.

The algorithm used to calculate new values for a graph is called a stat, short for statistical transformation.

We can plot the same plot as the previous one with stat_count().

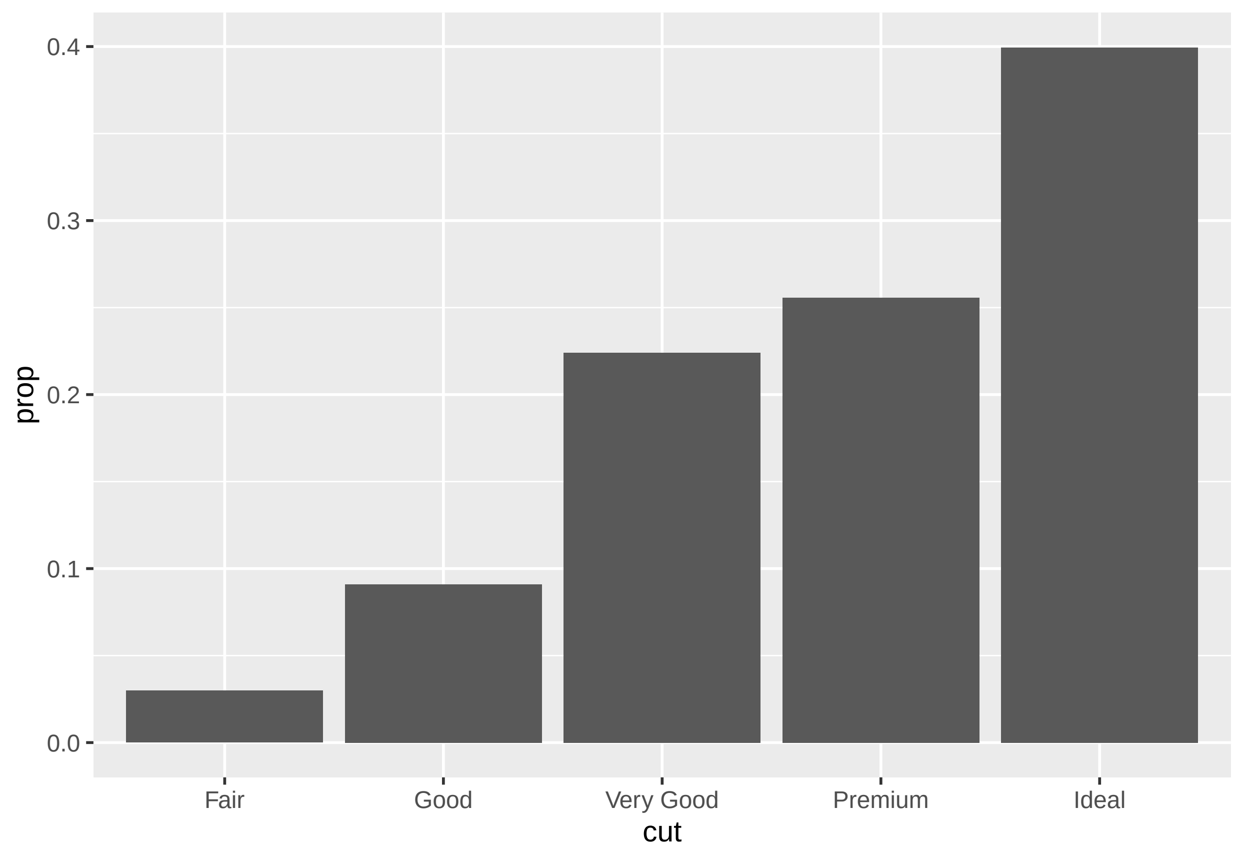
```
ggplot(data = diamonds) +
  stat_count(mapping = aes(x = cut))
```



This works because every geom has a default stat, and every stat has a default geom.

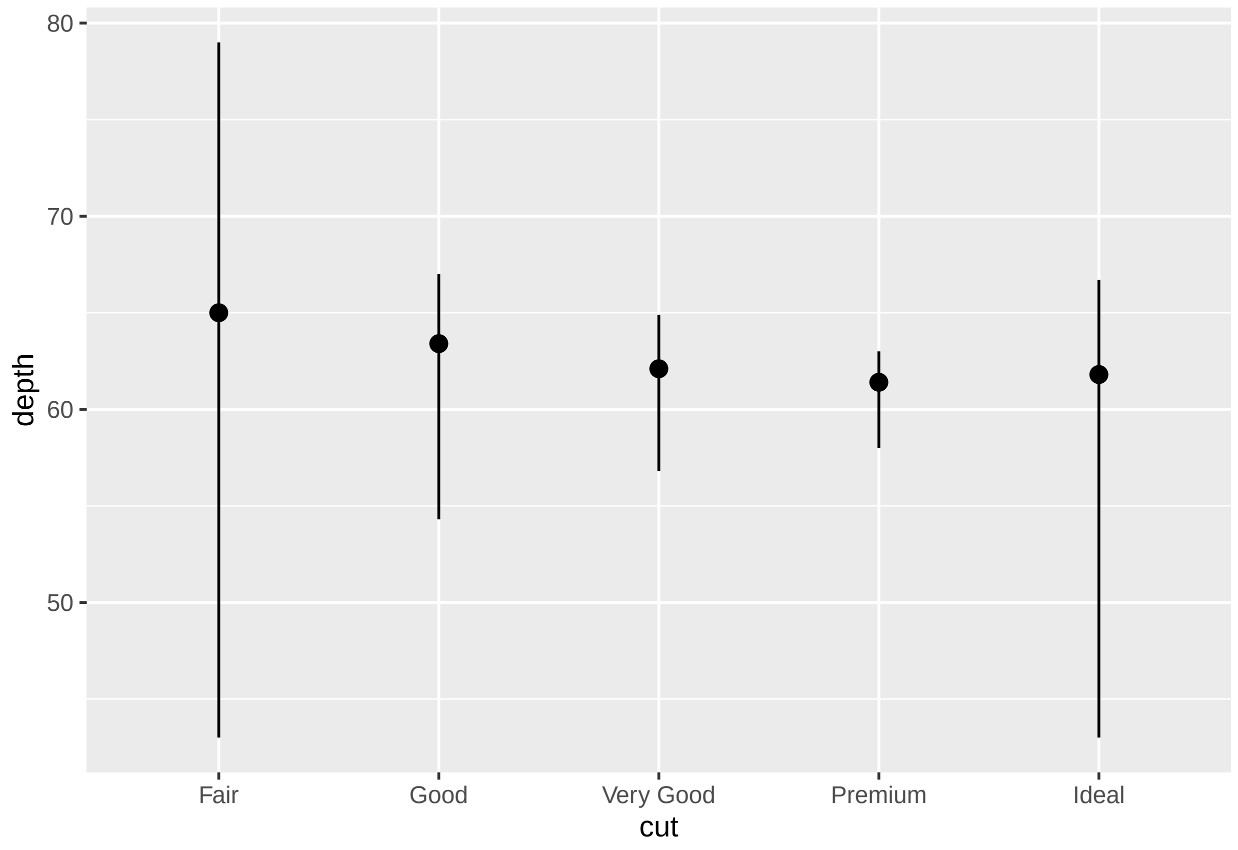
To display a bar chart of proportion, rather than count.

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, y = ..prop.., group = 1) )
```



stat_summary() summarizes the y values for each unique x value, to draw attention to the summary that is being computed.

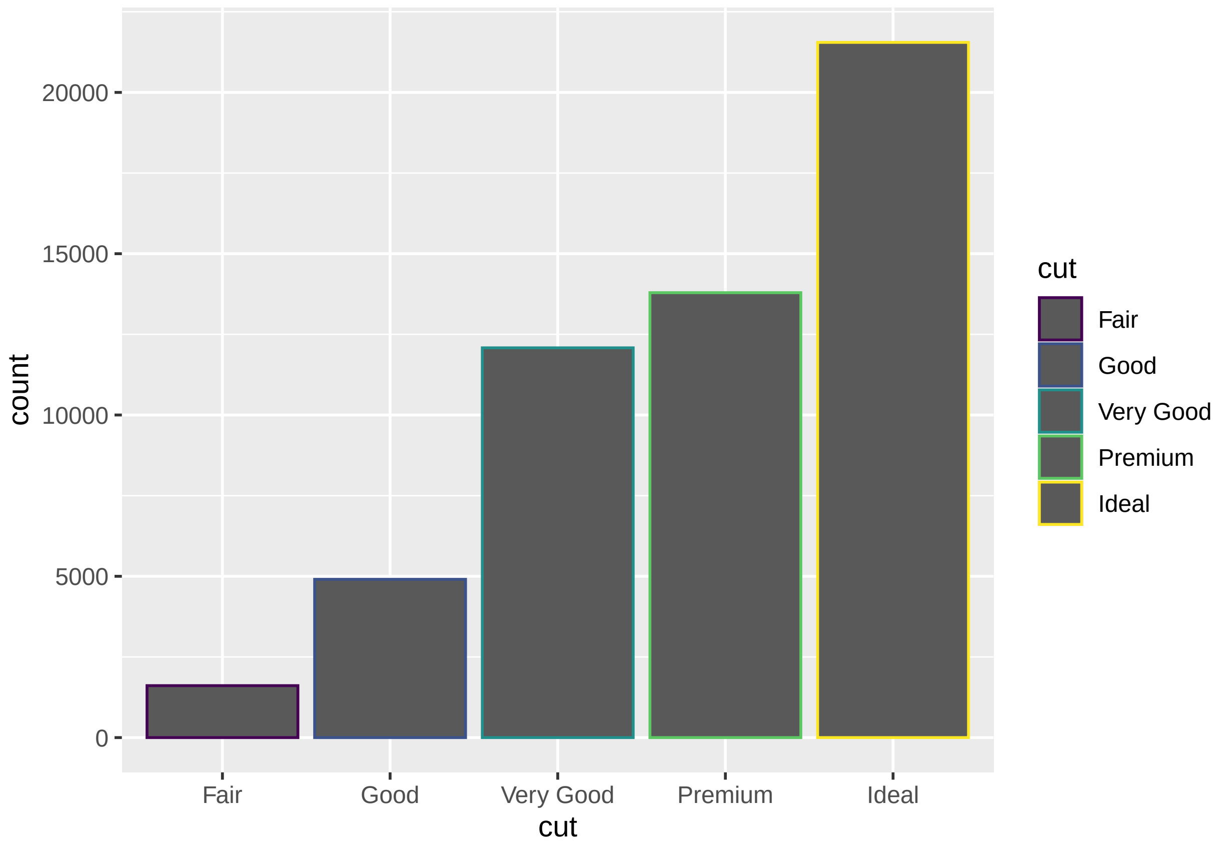
```
ggplot(data = diamonds) +
  stat_summary(mapping = aes(x = cut, y = depth), fun.min = min, fun.max = max, fun = median)
```



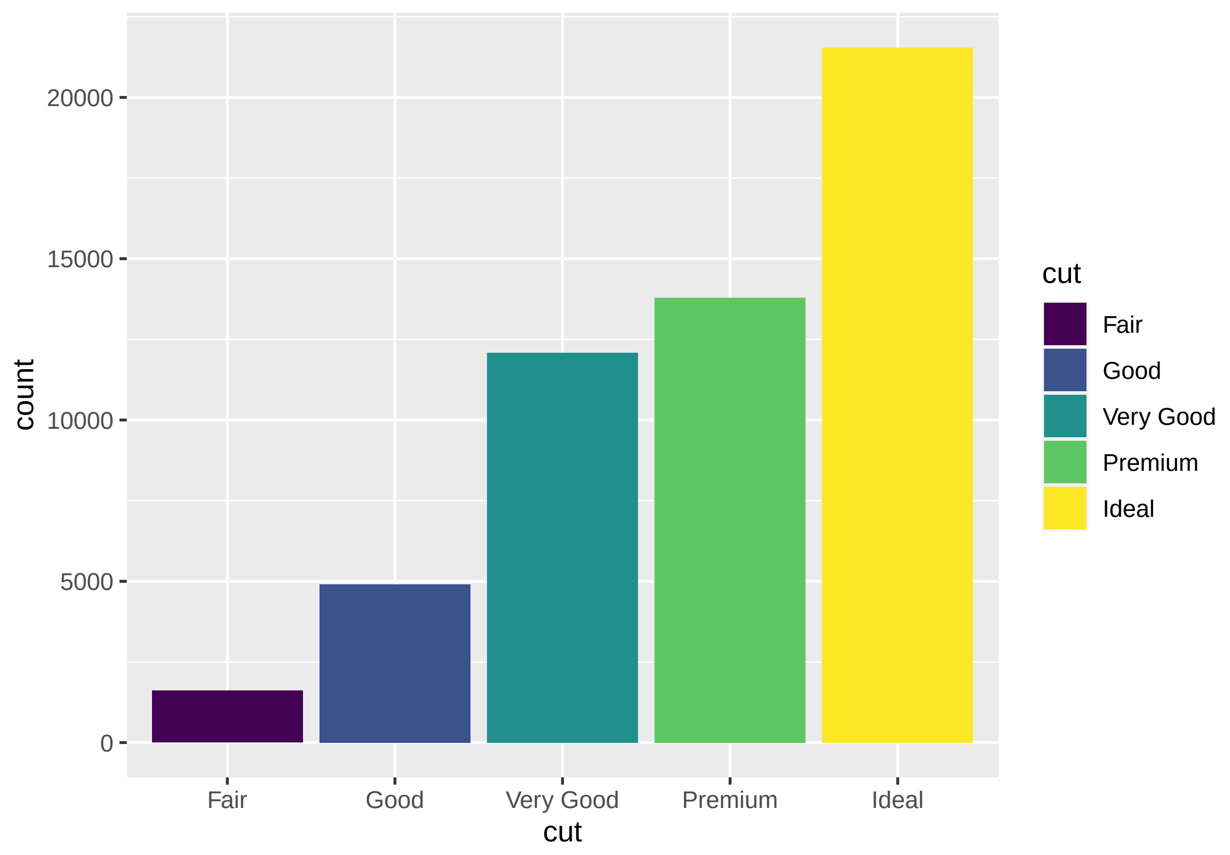
Position Adjustments

You can color a bar chart using either the color aesthetic, or more usefully, fill.

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

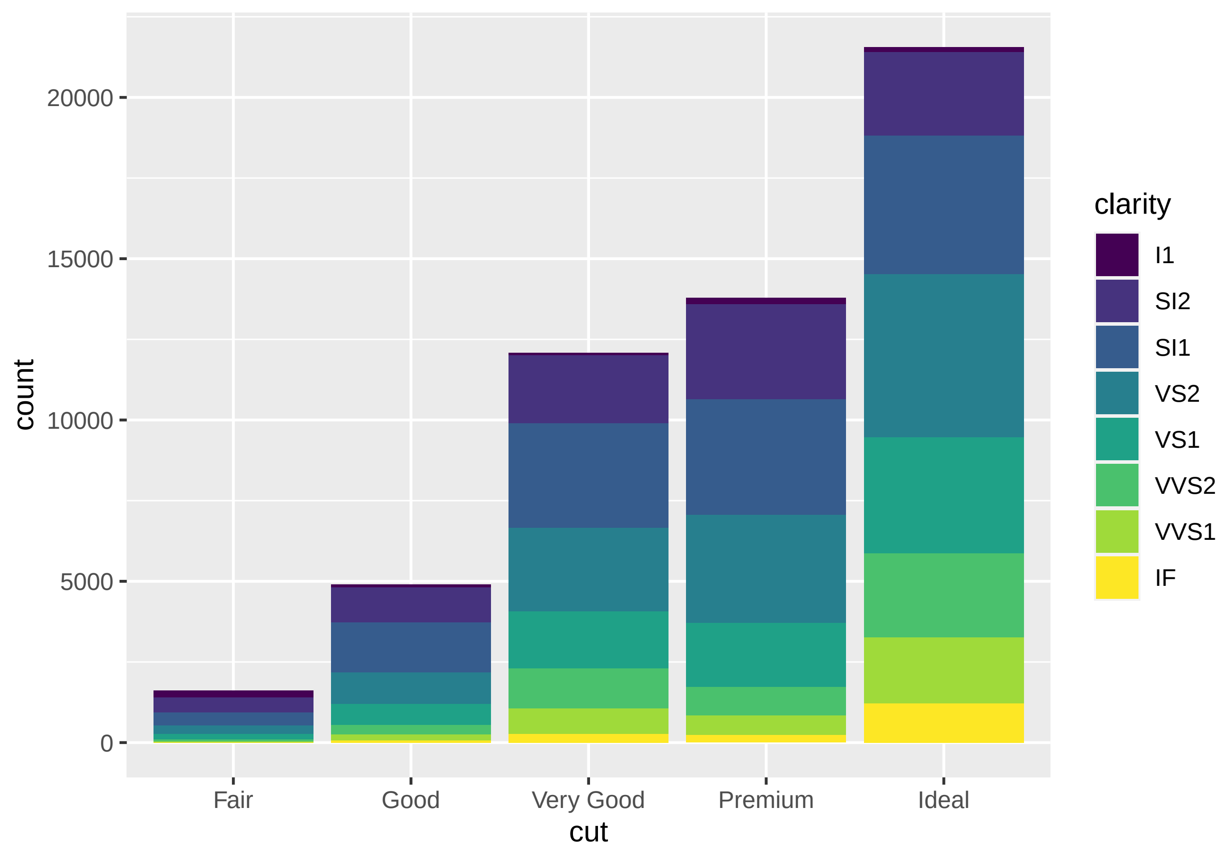


```r
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = cut))
```



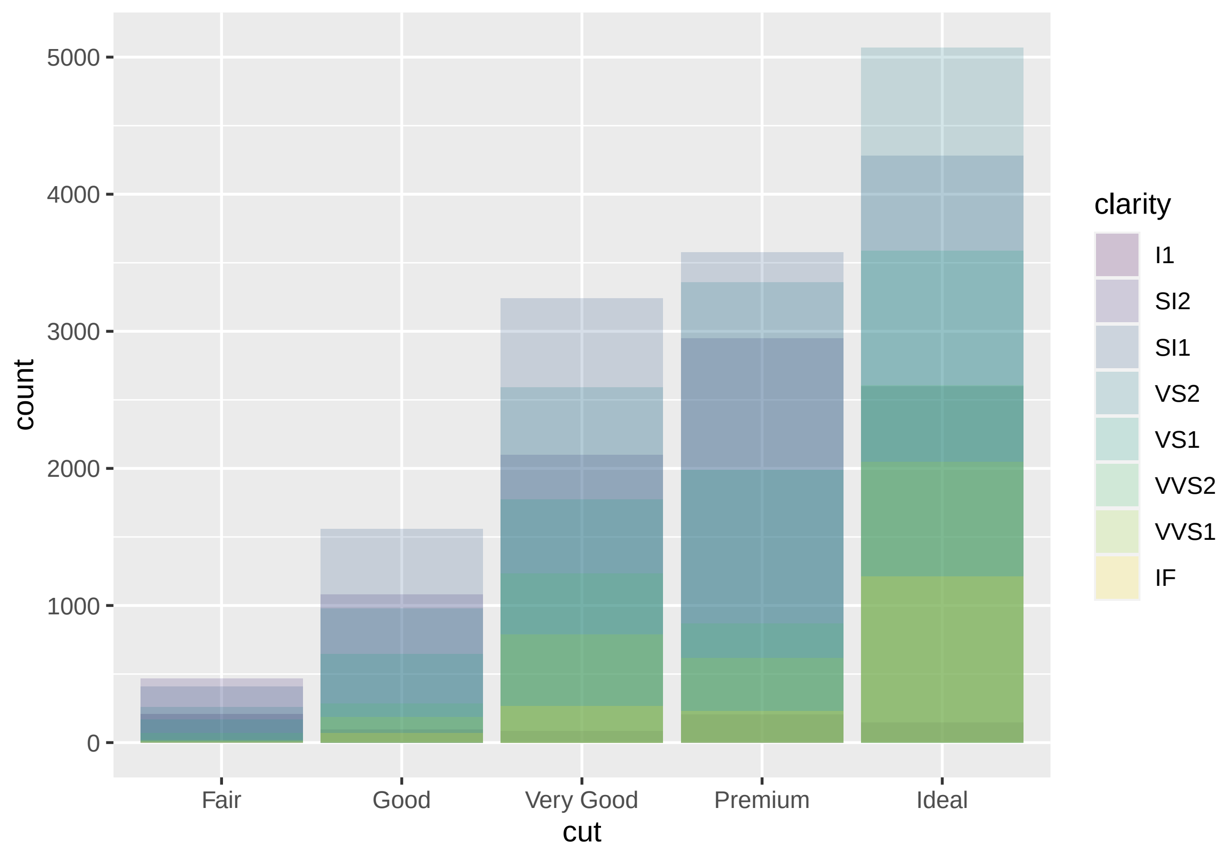
If you map the fill aesthetic to another variable, like clarity: the bars are automatically stacked. Each colored rectangle represents a combination of cut and clarity.

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

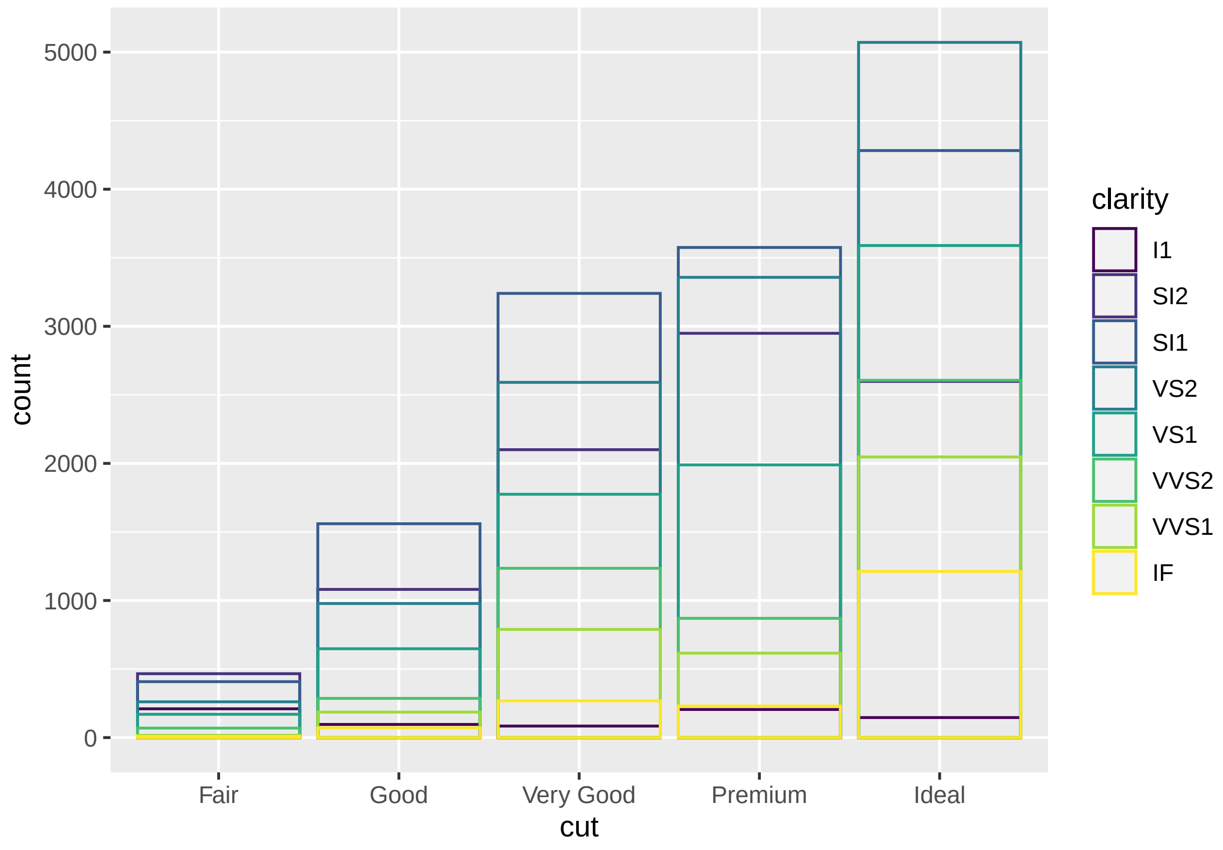


The stacking is performed automatically by the position adjustment specified by the position argument. If you don't want a stacked bar chart, you can use one of three other options: "identity", "dodge" or "fill".

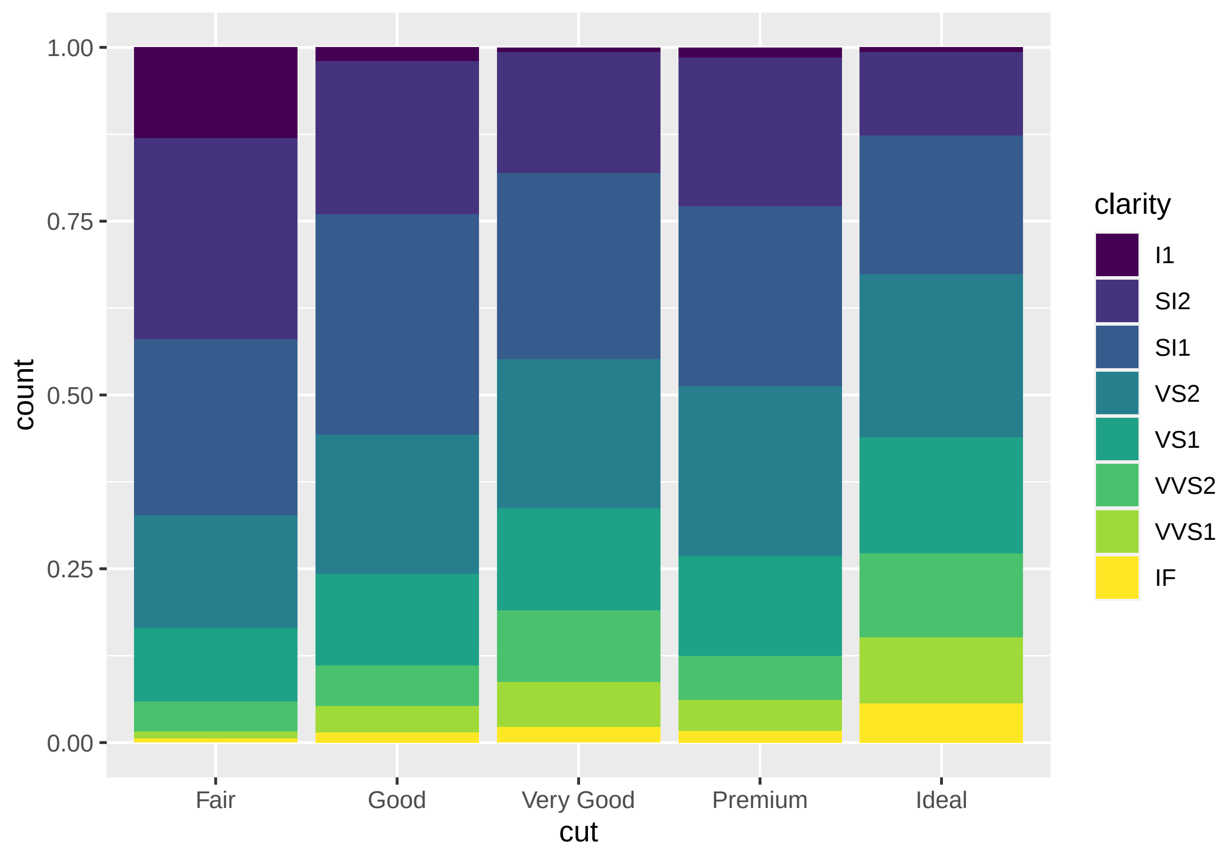
```
ggplot(data = diamonds, mapping = aes(x = cut, fill = clarity)) +
  geom_bar(alpha = 1/5, position = "identity")
```



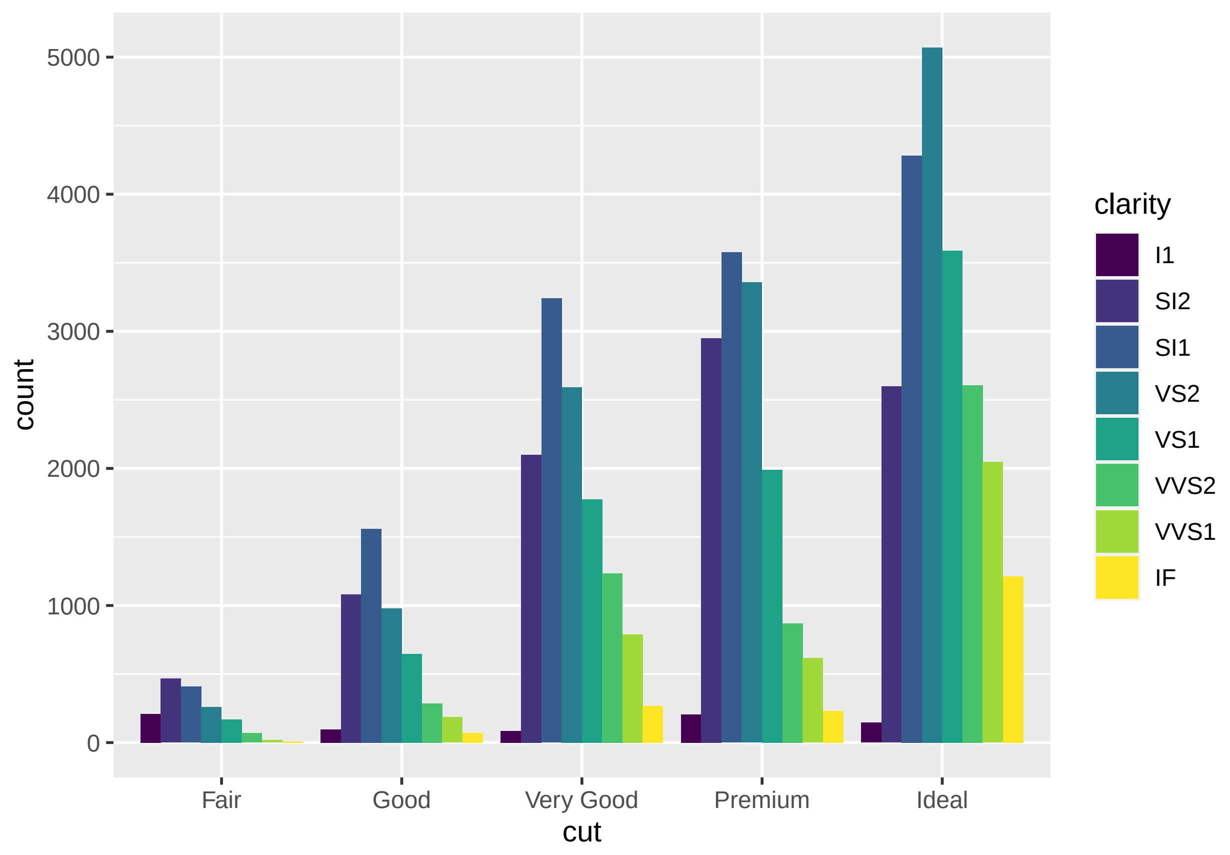
```r
ggplot(data = diamonds, mapping = aes(x = cut, color = clarity)) +
  geom_bar(fill = NA, position = "identity")
```



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

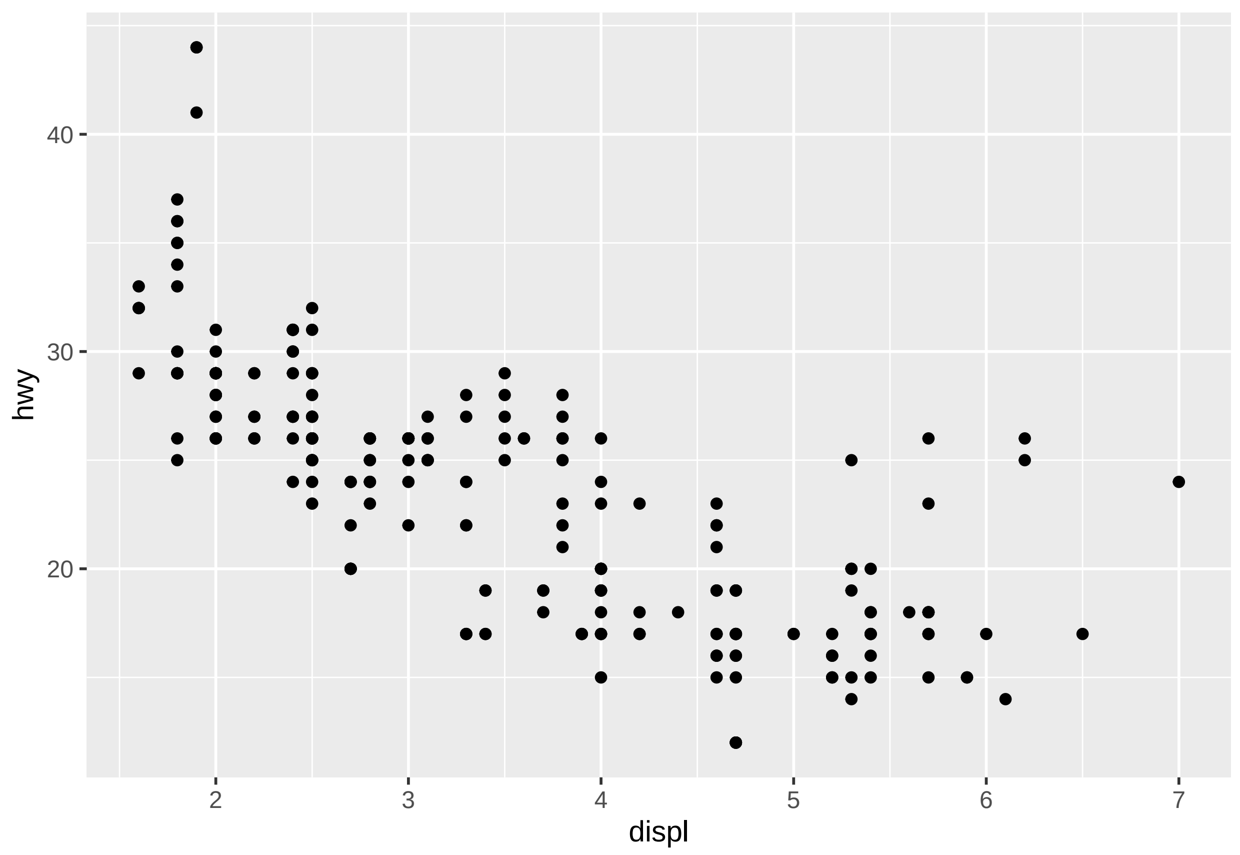


```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity), position = "dodge")
```

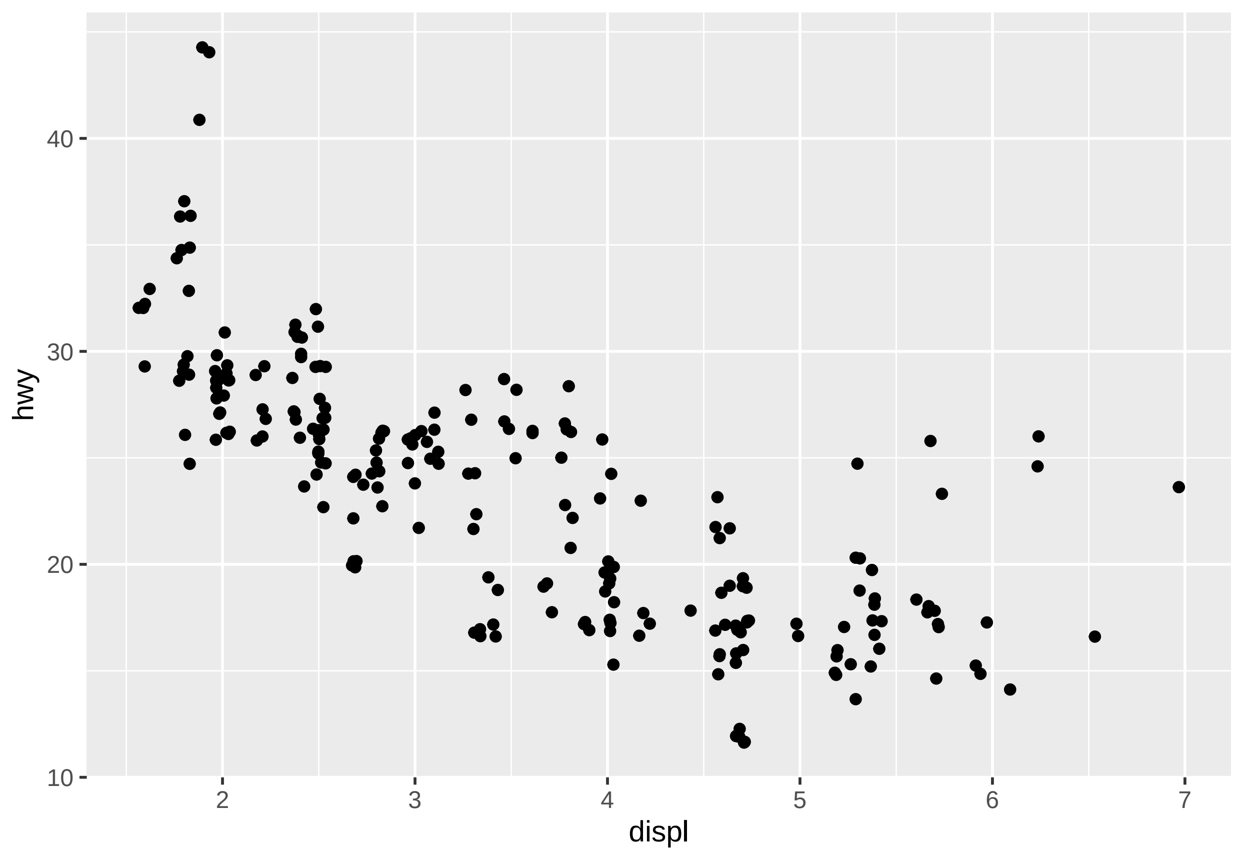


There's one other type of adjustment that's not useful for bar charts, but it can be very useful for scatterplots. On one of the previous plot was displaying only 126 points, even though there were 234 observations in the dataset. It is because the values of hwy and displ are rounded so the points appear on a grid and many points overlap each other. This problem is known as overplotting. You can avoid this gridding by setting the position adjustment to "jitter." position = "jitter" adds a small amount of random noise to each point. This spreads the points out because no two points are likely to receive the same amount of random noise.

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



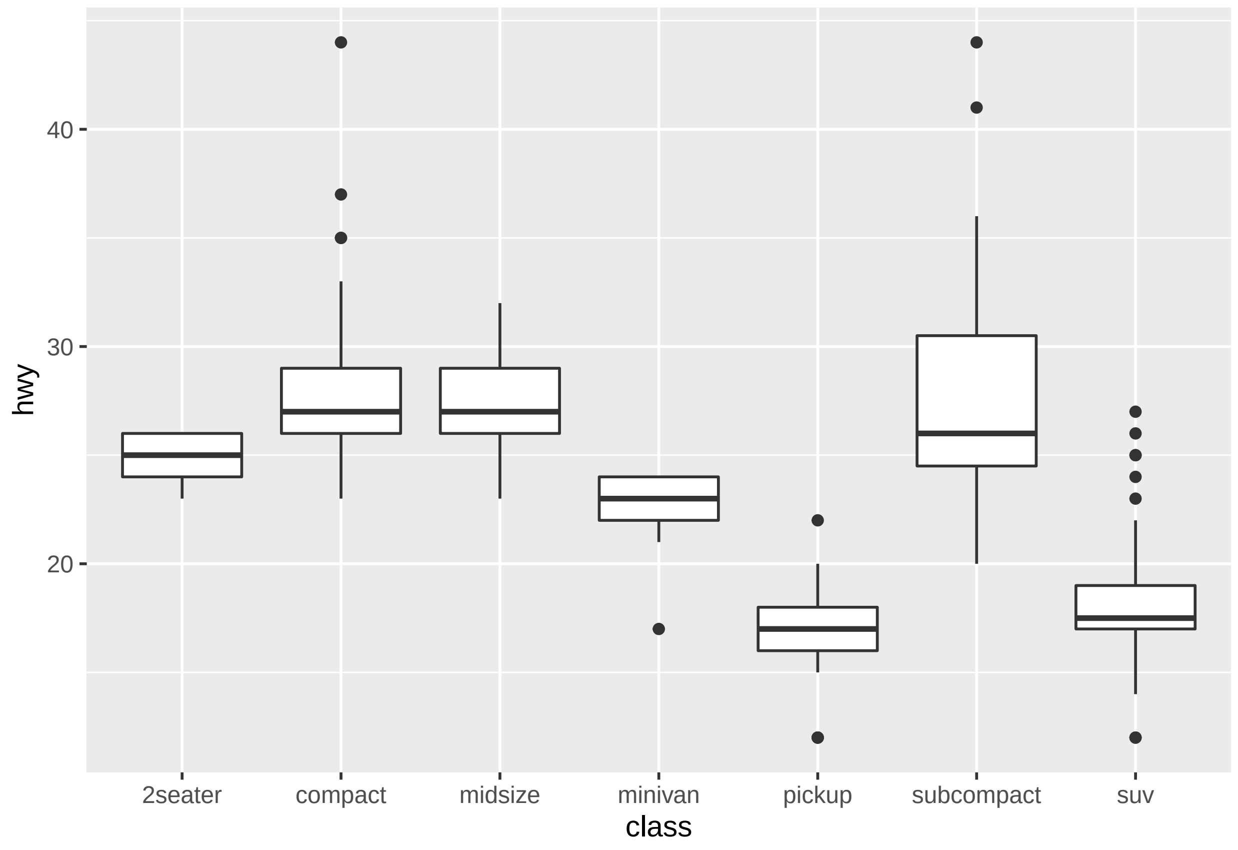
```r
ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y = hwy), position = "jitter")
```



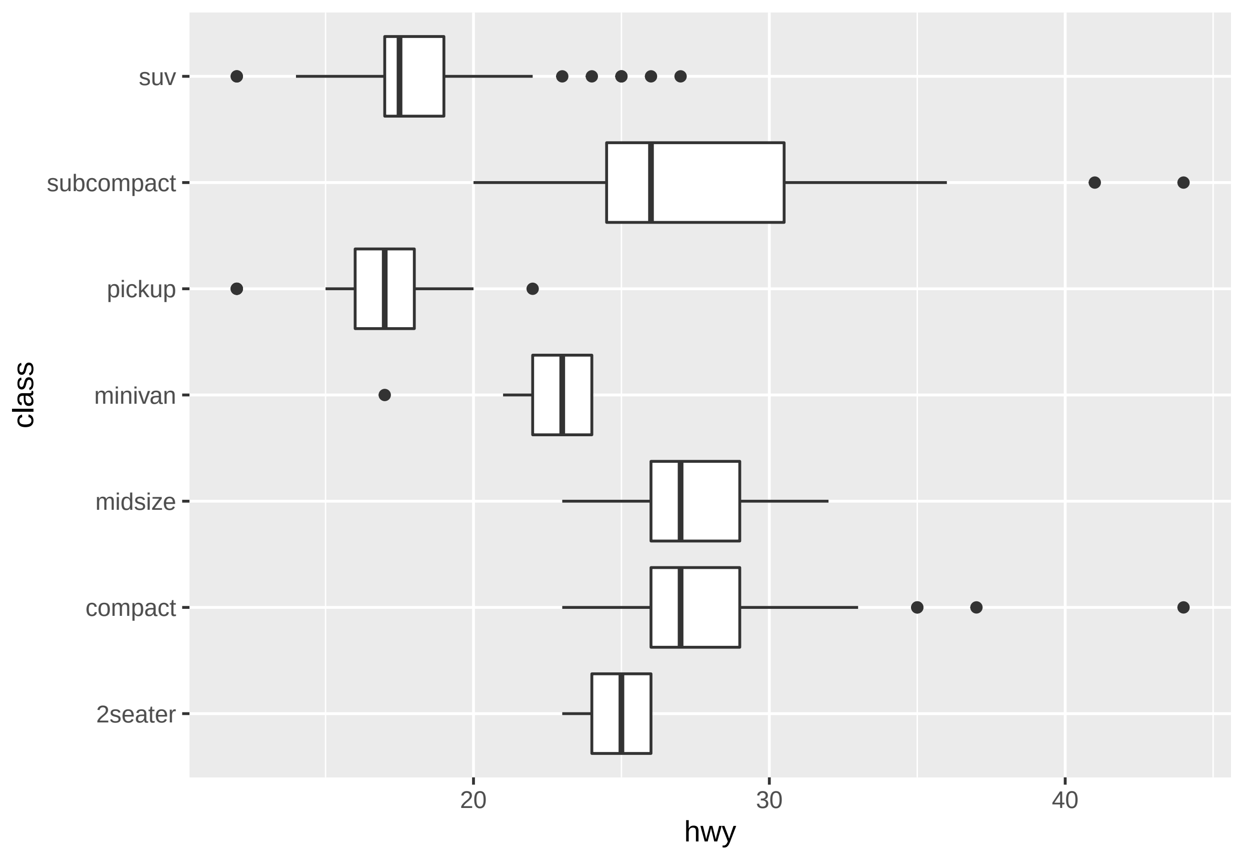
Coordinate Systems

coord_flip() switches the x- and y-axes.

```
ggplot(data = mpg, mapping = aes(x = class, y = hwy)) +
  geom_boxplot()
```



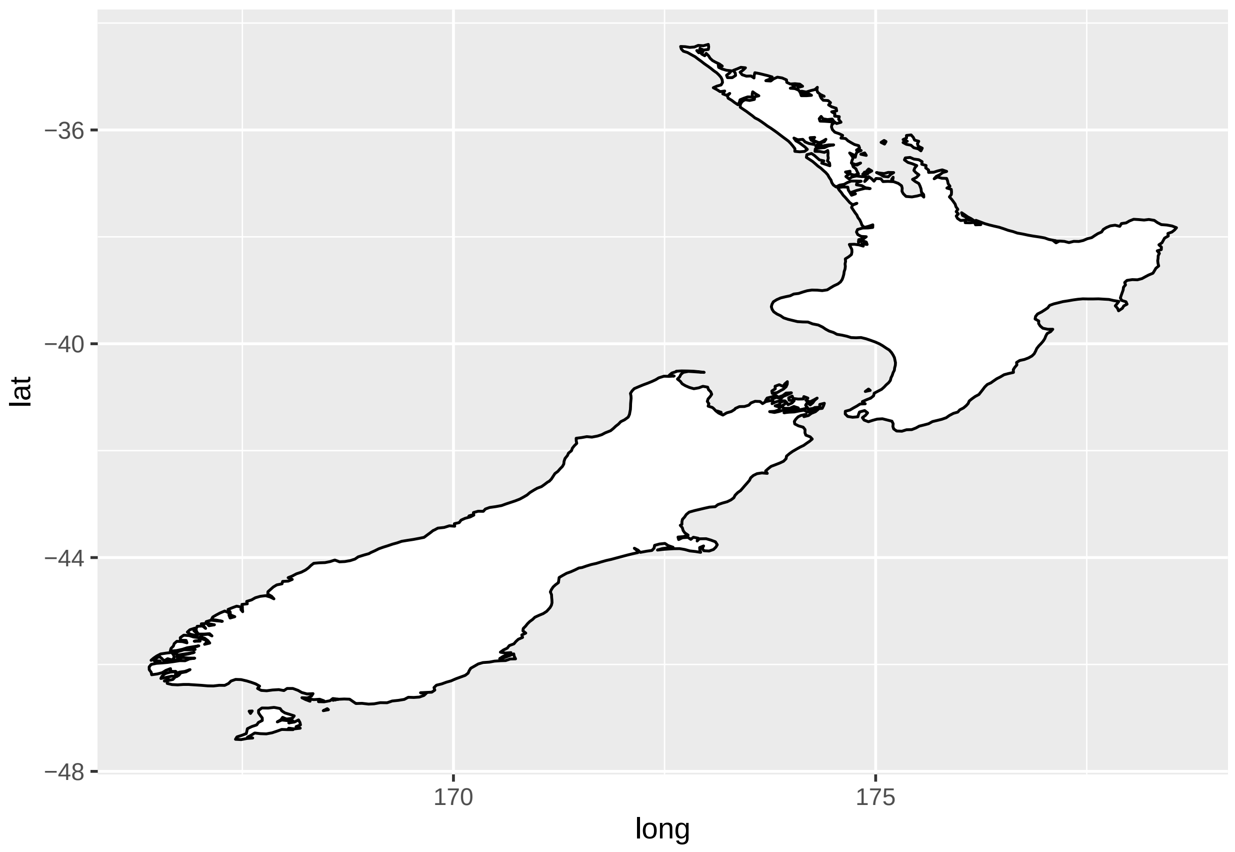
```
ggplot(data = mpg, mapping = aes(x = class, y = hwy)) +
  geom_boxplot() +
  coord_flip()
```



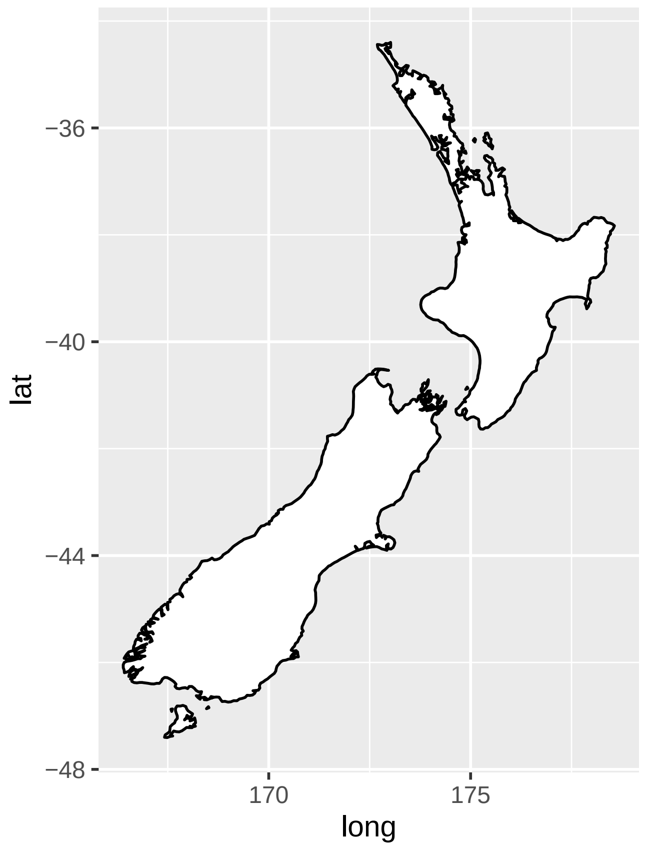
coord_quickmap() sets the aspect ratio correctly for maps.

```
nz <- map_data("nz")

ggplot(nz, aes(long, lat, group = group)) +
  geom_polygon(fill = "white", color = "black")
```



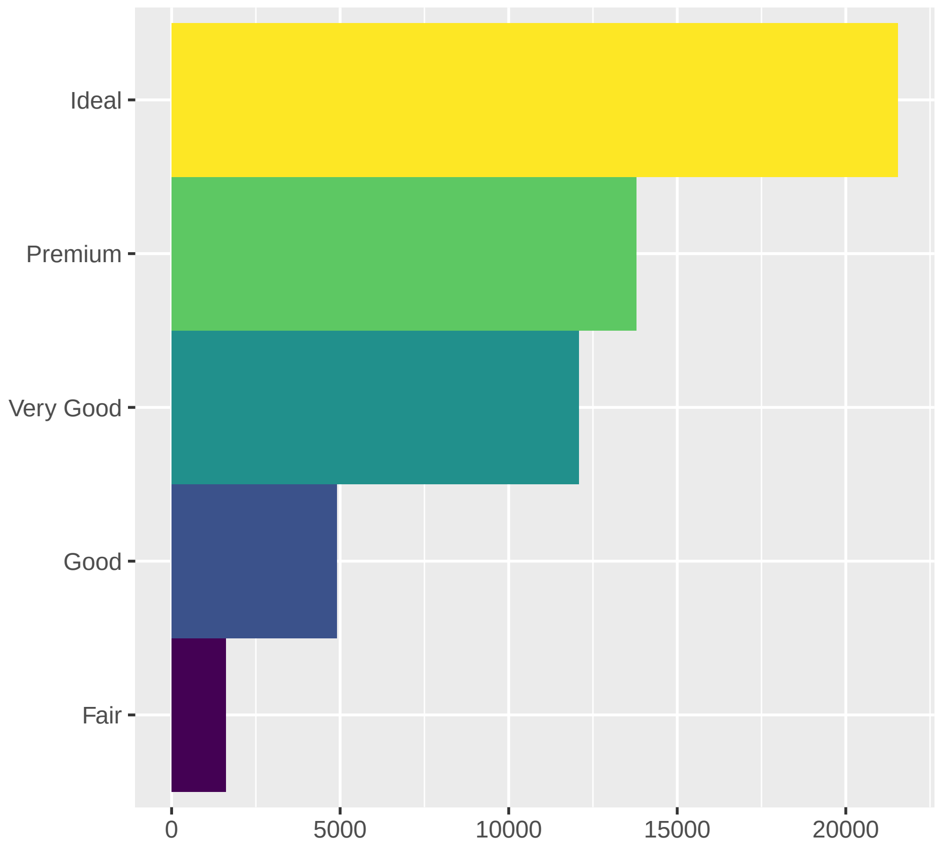
```
ggplot(nz, aes(long, lat, group = group)) +
  geom_polygon(fill = "white", color = "black") +
  coord_quickmap()
```



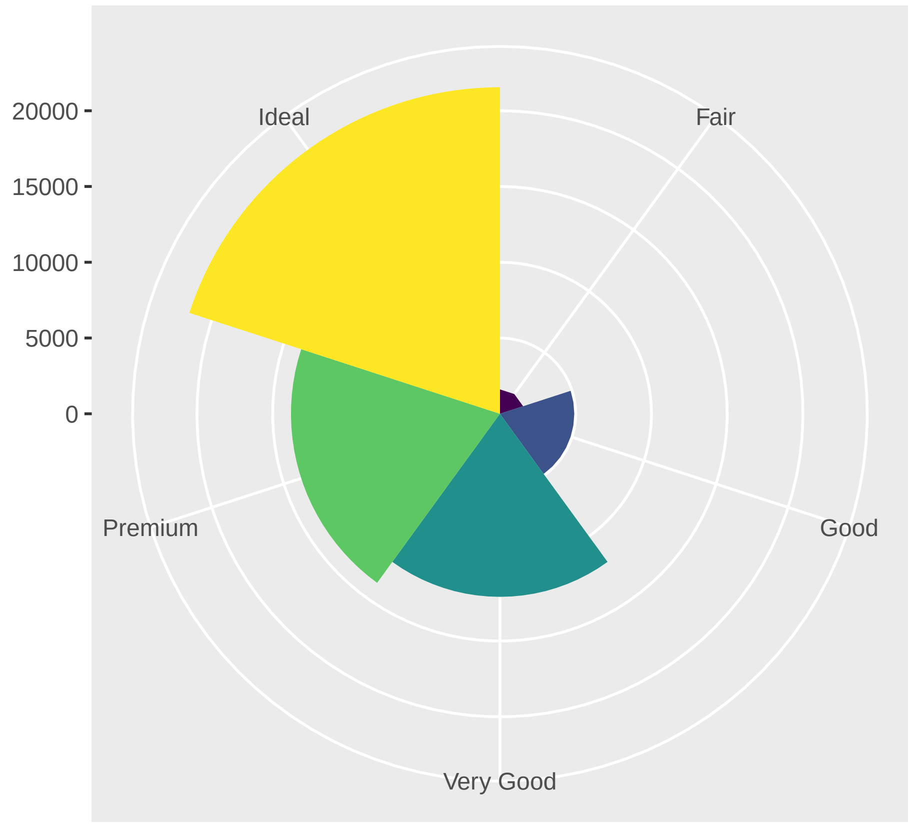
coord_polar() uses polar coordinates. Polar coordinates reveal an interesting connection between a bar chart and a Coxcomb chart.

```
bar <- ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = cut), show.legend = FALSE, width = 1) +
  theme(aspect.ratio = 1) +
  labs(x = NULL, y = NULL)

bar + coord_flip()
```



```
bar + coord_polar()
```


20000
15000
10000
5000
0
Ideal
Fair
Good
Very Good
Premium