

# Visualizing data

**PSY9219M & PSY9251M**

28/10/2025

# Data frames and tibbles

```
## # A tibble: 16 × 4
## # Groups:   Participant [8]
##   Participant Viewpoint B1RT B2RT
##   <int> <chr> <dbl> <dbl>
## 1         1 Different  499.  476.
## 2         1 Same      558.  417.
## 3         2 Different  449.  300.
## 4         2 Same      524.  539.
## 5         3 Different  432.  380.
## 6         3 Same      510.  383.
## 7         4 Different  522.  452.
## 8         4 Same      459.  383.
## 9         5 Different  505.  336.
## 10        5 Same      495.  387.
## 11        6 Different  453.  400.
## 12        6 Same      445.  410.
## 13        7 Different  567.  447.
## 14        7 Same      456.  339.
## 15        8 Different  489.  374.
## 16        8 Same      489.  445.
```

Data frames/tibbles are structured tables of data.

Each column contains data of the same basic type (i.e. a column can be numeric or character, but not both).

# Tidy data

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.



country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272015272
China	2000	216766	128042583

variables



country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272015272
China	2000	216766	128042583

observations



country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272015272
China	2000	216766	128042583

values

# Reshaping your data

The **tidyr** package has functions for *reshaping* data in order to make it *tidy*.

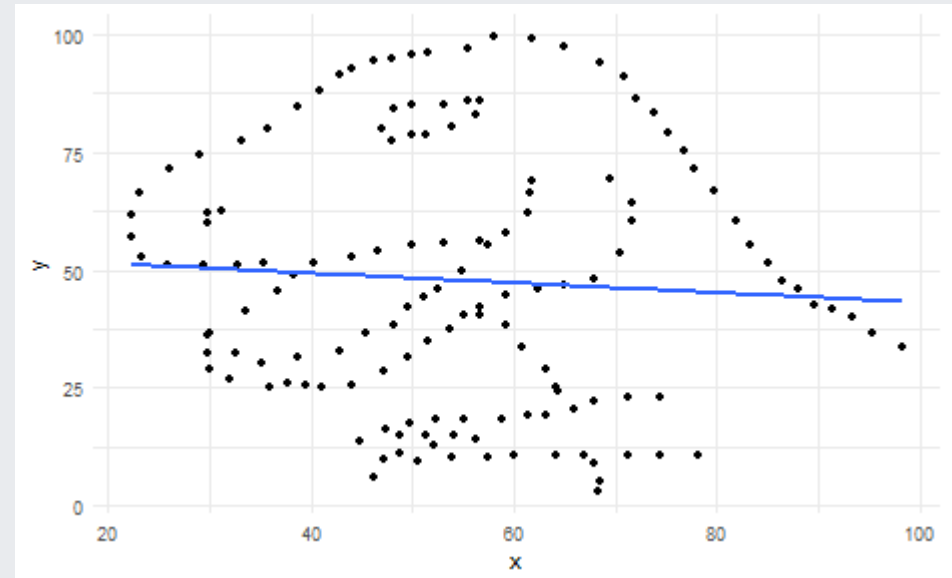
wide

id	x	y	z
1	a	c	e
2	b	d	f

# Visualizing data

# Why visualize data?

1. Graphs help you rapidly examine the structure of the data.
2. Graphs help you communicate the important statistical features of data.
3. It's often easier to spot unexpected issues using graphs than staring at a bunch of numbers.



# Getting a quick look at your data

Plotting helps you quickly gain an understanding of the structure of your data.

Here's some recent data about the UK's prison population.

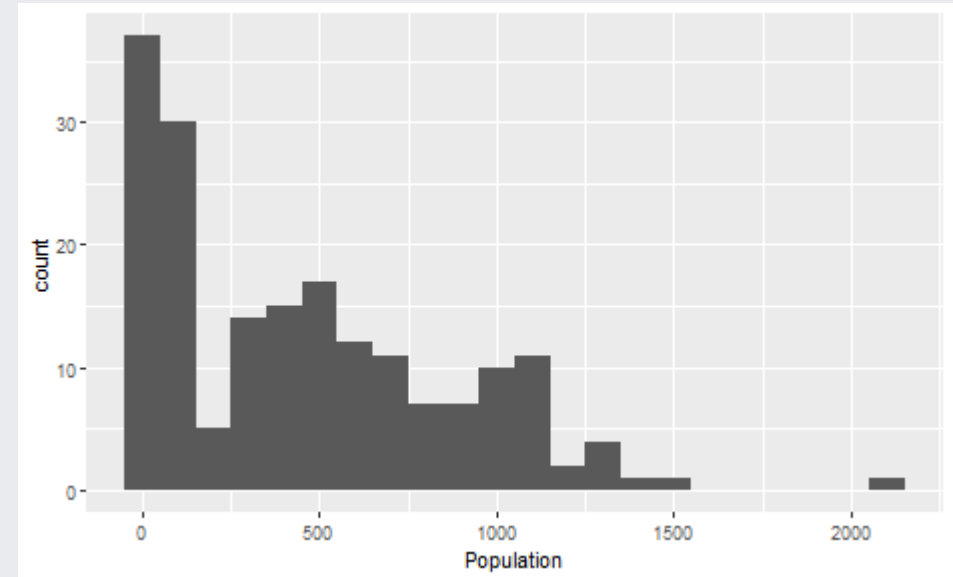
```
## # A tibble: 22,409 × 6
##   View                                Date      Establishment Sex    Age / Cu...1 Popul...2
##   <chr>                                <chr>      <chr>          <chr>   <chr>          <dbl>
## 1 a Establishment*Sex*Age Group 2015-06 Altcourse      Male   Adults (2...    922
## 2 a Establishment*Sex*Age Group 2015-06 Altcourse      Male   Juveniles...   169
## 3 a Establishment*Sex*Age Group 2015-06 Ashfield       Male   Adults (2...   389
## 4 a Establishment*Sex*Age Group 2015-06 Askham Grange Female Adults (2...    NA
## 5 a Establishment*Sex*Age Group 2015-06 Askham Grange Female Juveniles...    NA
## 6 a Establishment*Sex*Age Group 2015-06 Aylesbury       Male   Adults (2...   113
## 7 a Establishment*Sex*Age Group 2015-06 Aylesbury       Male   Juveniles...   268
## 8 a Establishment*Sex*Age Group 2015-06 Bedford         Male   Adults (2...   459
## 9 a Establishment*Sex*Age Group 2015-06 Bedford         Male   Juveniles...    30
## 10 a Establishment*Sex*Age Group 2015-06 Belmarsh        Male   Adults (2...   794
## # ... with 22,399 more rows, and abbreviated variable names
## #   1`Age / Custody / Nationality / Offence Group`, 2Population
```

# Getting a quick look at your data

Let's look at the UK prison population as of December 2017, split by establishment, sex, and age group.

First we filter out all but the rows I'm interested in. Don't worry about understanding this code... (yet!)

```
pris_pop %>%  
  filter(View == "a Establishment*Sex*Age C  
         Date == "2017-12") %>%  
  ggplot(aes(x = Population)) +  
  stat_bin(binwidth = 100)
```





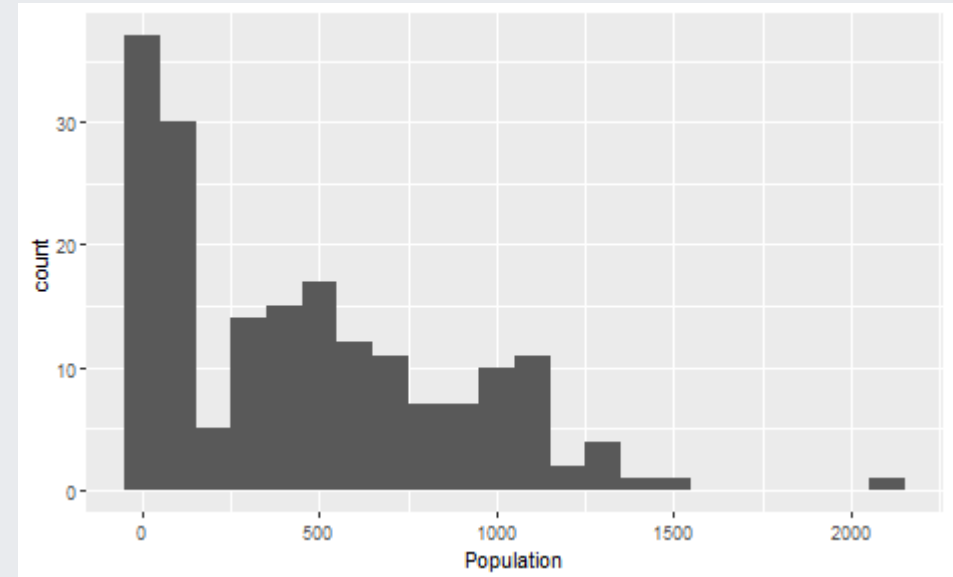
# Getting a quick understanding of your data

This is a histogram showing the distribution of prison populations in bins of 100 inmates.

Some obvious features:

1. The data is heavily skewed - lots of small values, few large values.
2. There may be a mixture of distributions - there's a big peak in the low numbers, then a dip, then a broader peak.

These two features suggest that there may be some structure we're missing with this plot.

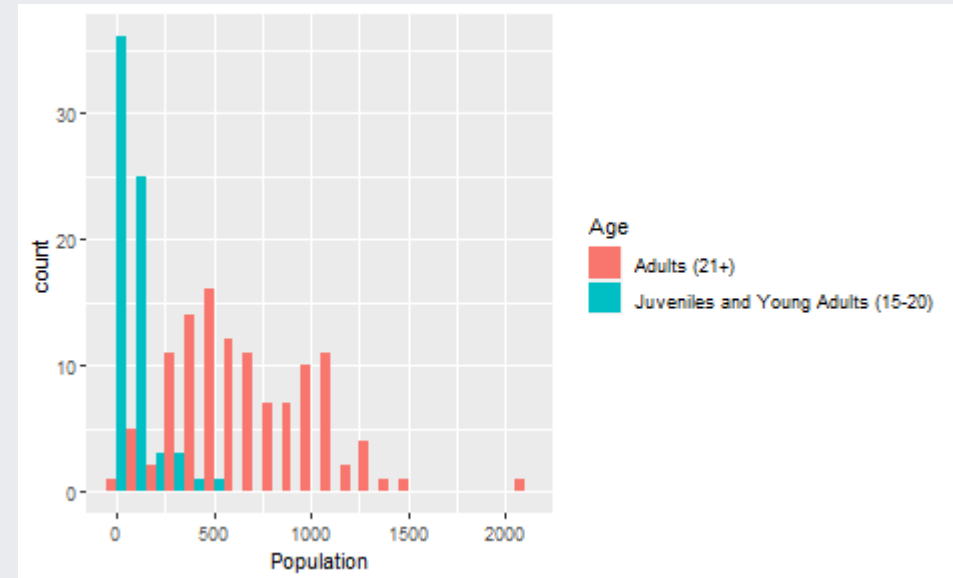


# Getting a quick understanding of your data

In the data, age is coded into "Juveniles and Young Adults (15-20)" and "Adults (21+)".

Let's see if Age underlies some of the features of the first plot.

```
pris_pop %>%  
  filter(View == "a Establishment*Sex*Age C  
         Date == "2017-12") %>%  
  ggplot(aes(x = Population,  
             fill = `Age / Custody / Nation  
stat_bin(binwidth = 100,  
         position = "dodge") +  
  labs(fill = "Age")
```



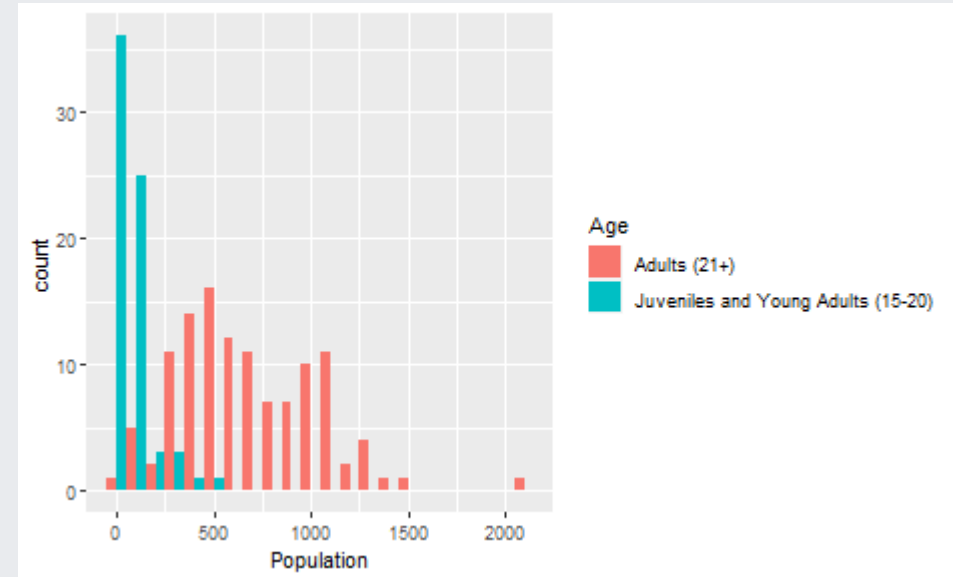
# Getting a quick understanding of your data

The "Juvenile" prison population underlies the lower peak.

Typically there are fewer than 200 juveniles in a given institution.

In addition, there are far fewer juveniles in prison than adults.

Note that while many institutions hold both adults and juveniles, some hold only adults and some hold only juveniles.

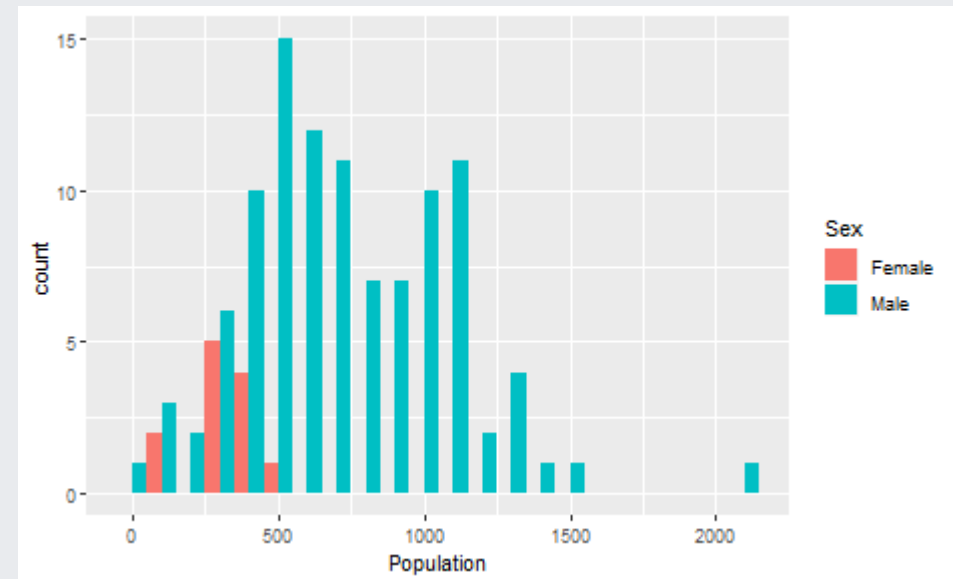


# Getting a quick understanding of your data

How do prison populations vary between men and women?

Here we focus on adults, excluding juveniles from the plot.

```
pris_pop %>%  
  rename(Age = `Age / Custody / Nationality`)  
  filter(View == "a Establishment*Sex*Age C",  
         Date == "2017-12",  
         Age == "Adults (21+)") %>%  
  ggplot(aes(x = Population,  
             fill = Sex)) +  
  stat_bin(binwidth = 100,  
          position = "dodge")
```

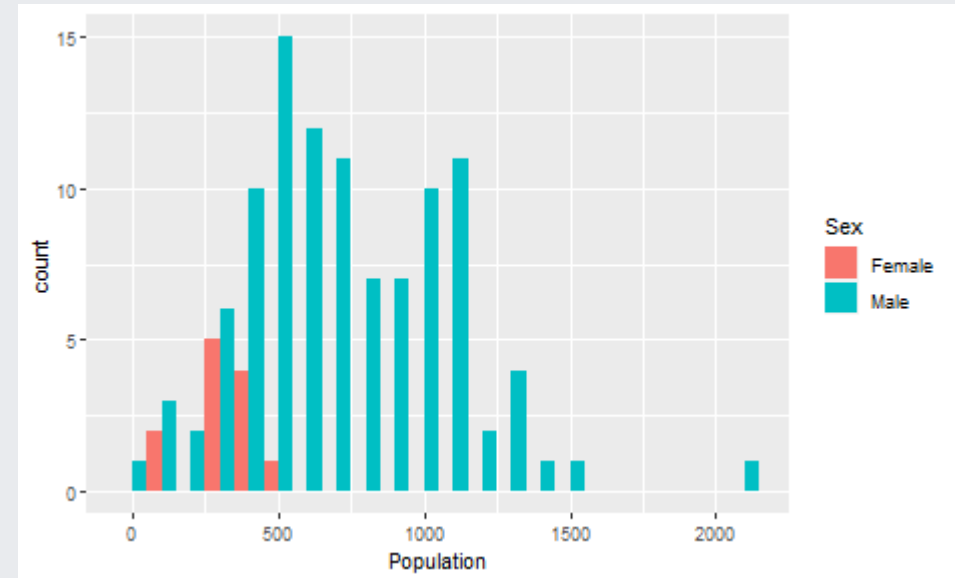


# Getting a quick understanding of your data

We can clearly see that there are far more men in prison than women.

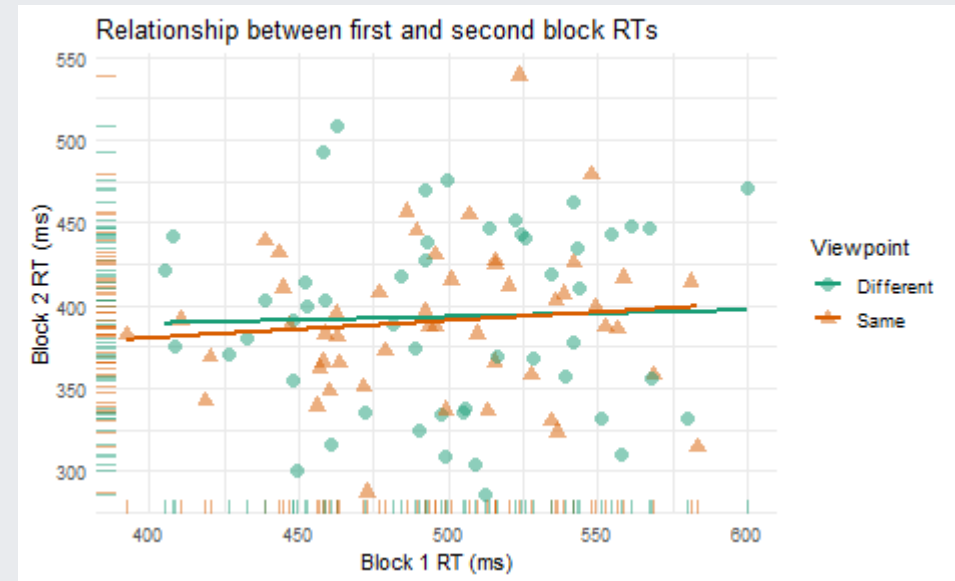
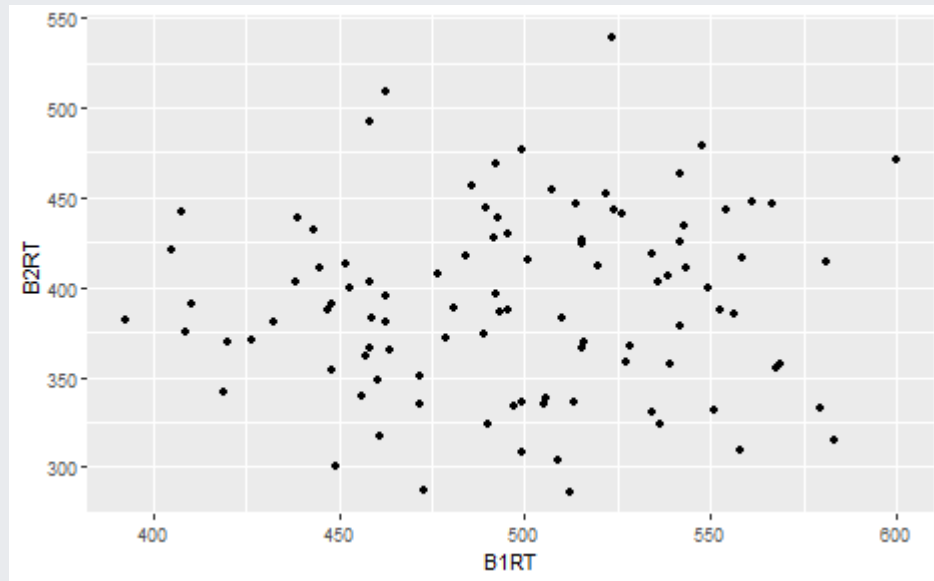
There are also far fewer institutions that hold women than institutions that hold men.

Also there are generally more men in any given institution than there are women.

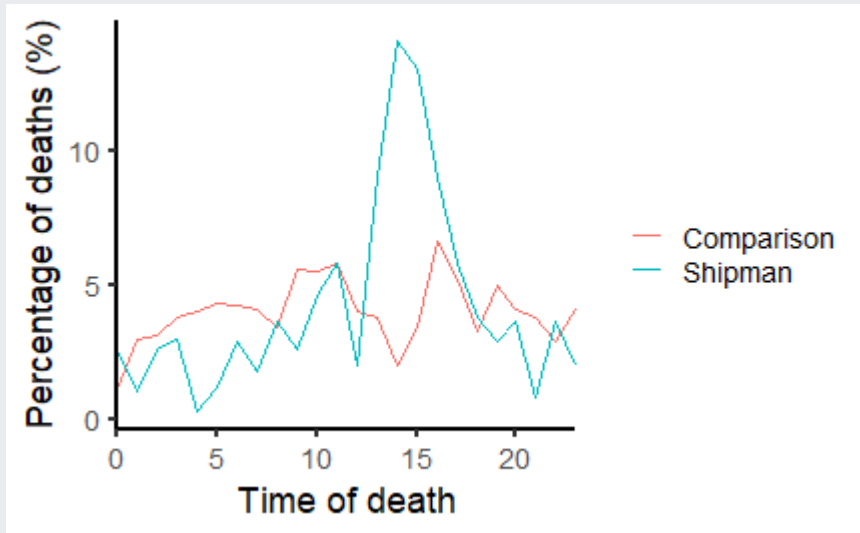


# Communicating your results

Plots are also useful for showing the statistical patterns in your data to go along with statistical tests.



# Communicating patterns



Strikingly different to similar GPs, many of Harold Shipman's patients died at a particular time of day.

A pattern like this passes the "inter-ocular trauma" test...

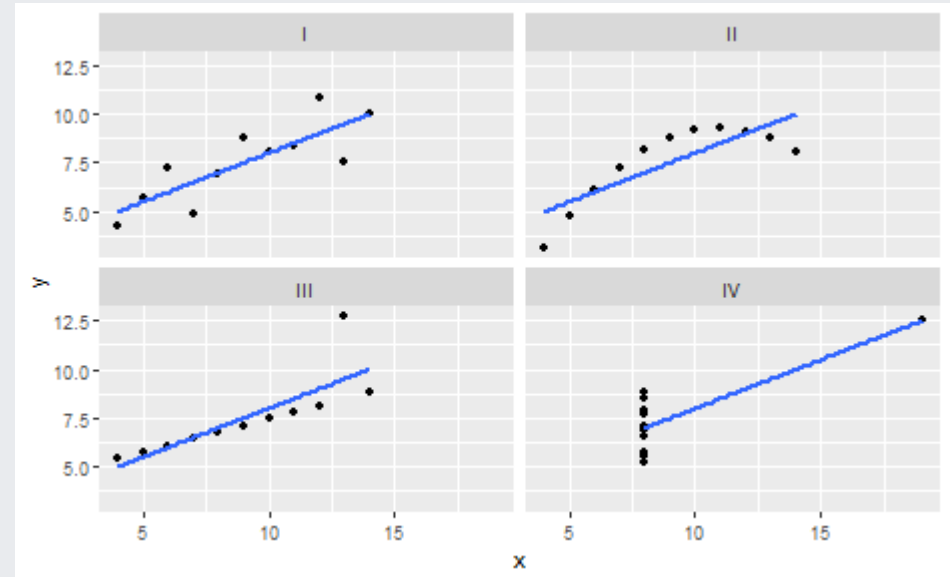
Spiegelhalter (2019), *The Art of Statistics*

# Spotting problems in your data

## Anscombe's Quartet

Every one of these plots shows sets of data with the same means, standard deviations, and correlation coefficients.

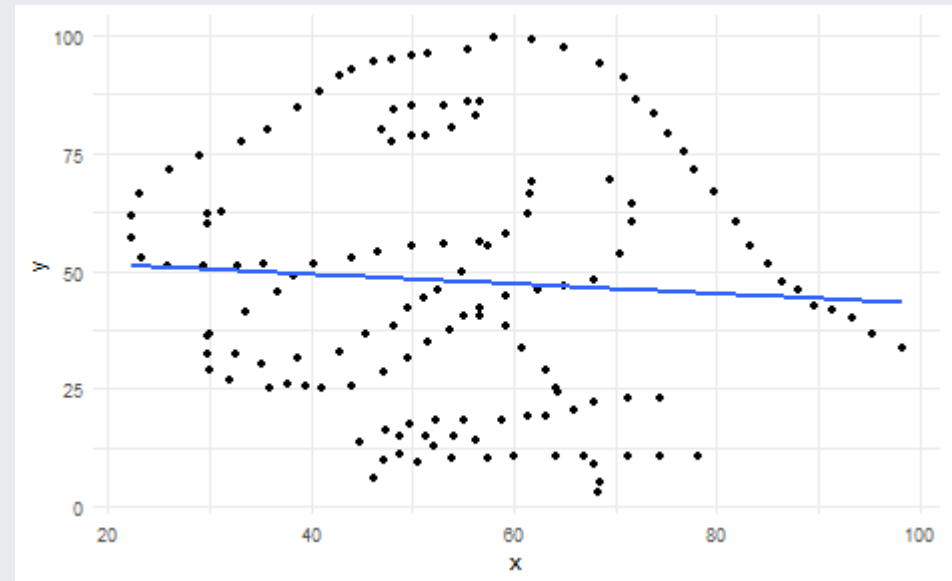
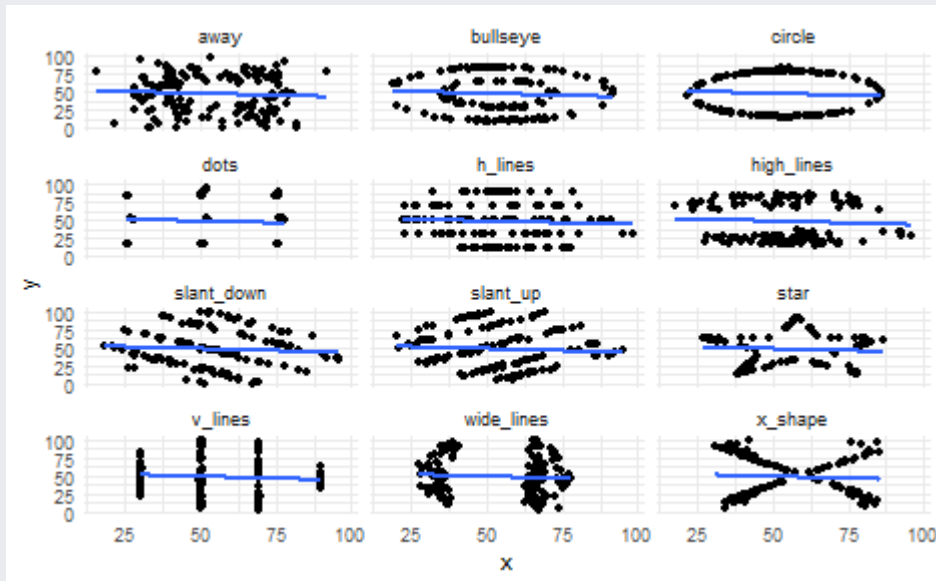
One is non-linear, one has an outlier, and one should have a categorical x-axis!





# Spotting problems in your data

## The Datasaurus Dozen



# The Grammar of Graphics



# ggplot2



**ggplot2** is one of the **tidyverse** packages.

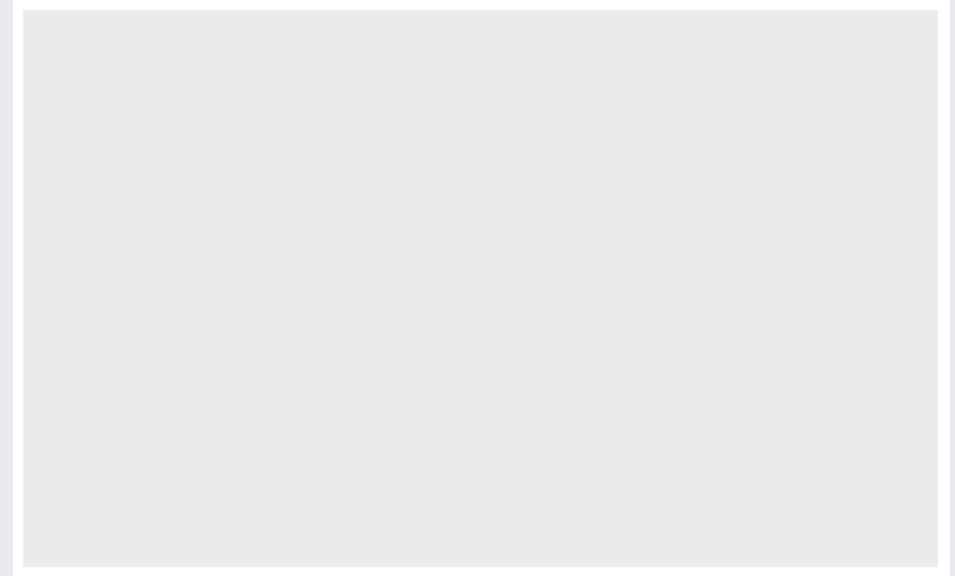
GG stands for the *Grammar of Graphics*.

The Grammar of Graphics is a principled approach to building plots from a few underlying structures:

1. A dataset
2. A coordinate system
3. *Geoms* (geometric shapes such as bars or points)

We begin with a blank canvas:

```
ggplot()
```



# The mpg dataset



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...
## # ... with 224 more rows
```

# Datasets and aesthetics



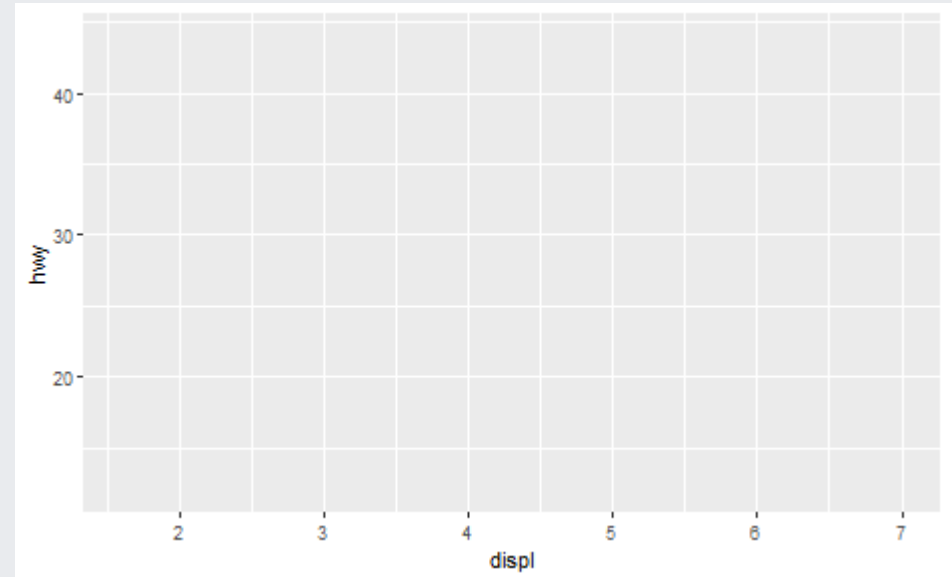
The first step is to add dataset and define some *aesthetics*.

Aesthetics are how we map elements of the data to parts of the plot.

The first two arguments to `ggplot()` are data and mapping.

We use the `aes()` function within this to map columns from the data to properties of the plot.

Here we use the 'displ' and 'hwy' columns from the *mpg* dataset to set up our co-ordinate system.



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

# Geoms and layers

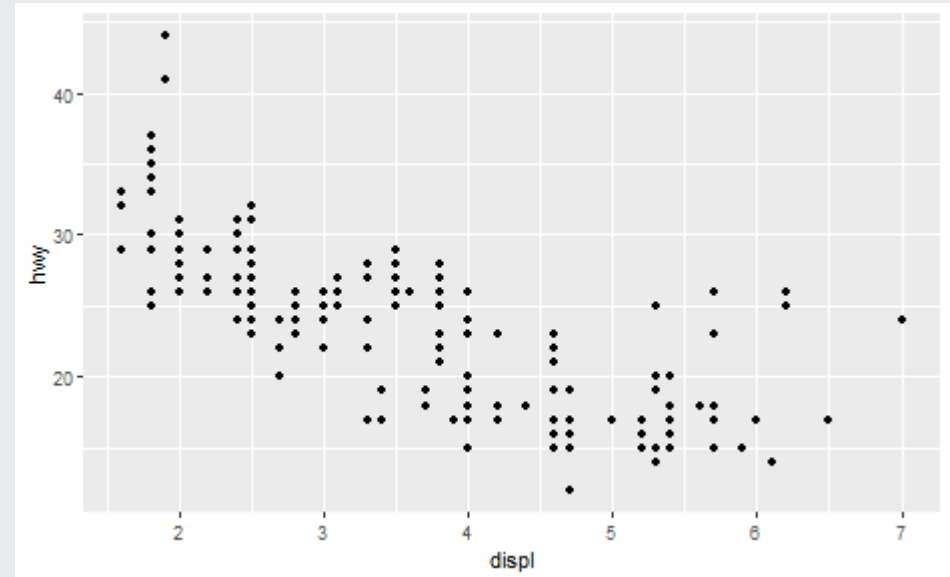


**geoms** are the geometric shapes we want to use to represent our data.

We add a new layer to our initial canvas using `+`, and then use one of the many `geom_*` functions to draw shapes on the new layer.

For a scatterplot, add a new layer using `geom_point()`.

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



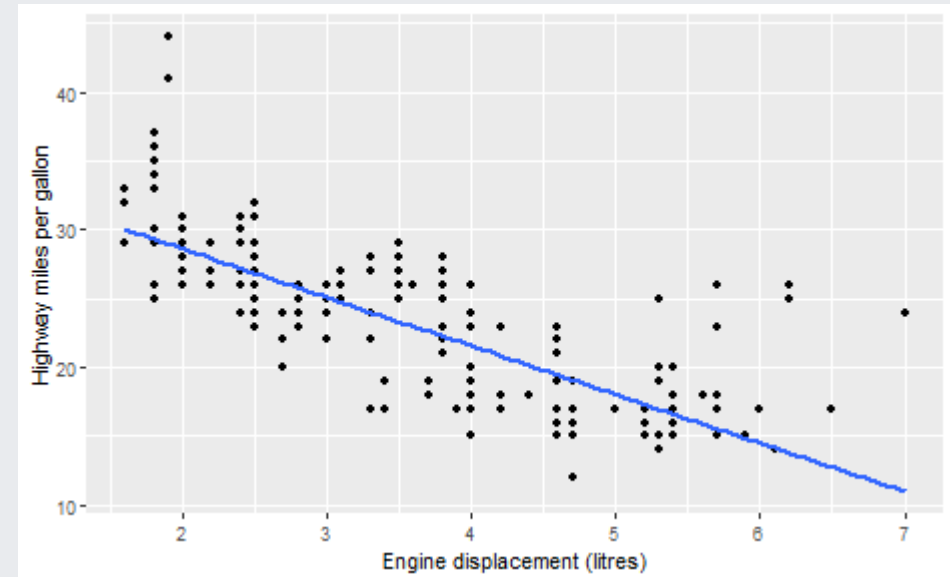
# Adding a linear model



A question we're pondering is what is the relationship between the variables on x- and y-axes?

We can add a linear regression line using `geom_smooth()` and specifying "lm" (linear model) for the argument method.

```
ggplot(data = mpg,
       mapping = aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(x = "Engine displacement (litres)",
       y = "Highway miles per gallon")
```



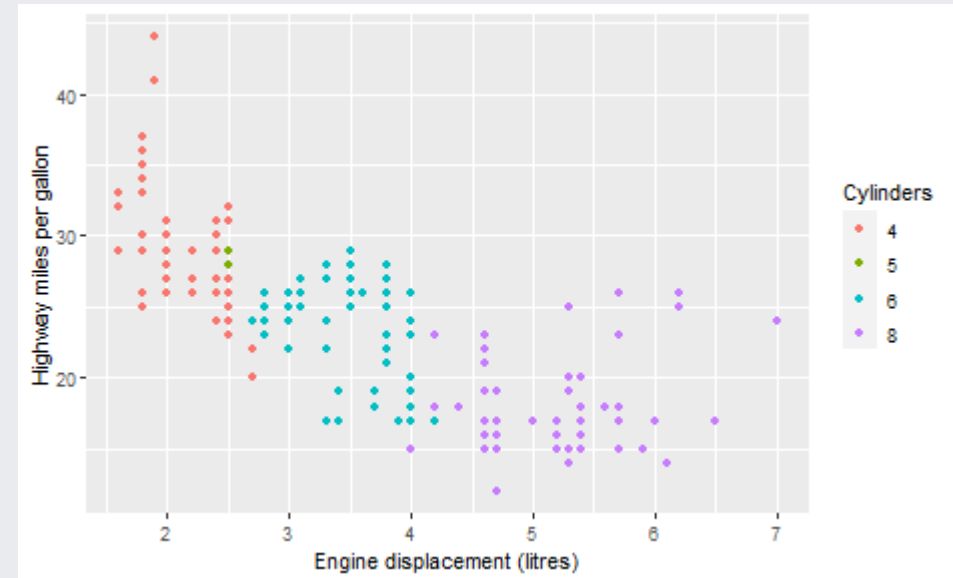
# Identifying groups



Another variable we know about is the number of cylinders in the engines - the *cyl* column.

*cyl* only has four unique levels, so it's best treated as a categorical variable and converted to a factor using `factor()`. Here, we use colour to identify different levels of *cyl*.

```
ggplot(data = mpg,
       mapping = aes(x = displ,
                     y = hwy,
                     colour = factor(cyl)))
  geom_point() +
  labs(x = "Engine displacement (litres)",
       y = "Highway miles per gallon",
       colour = "Cylinders")
```



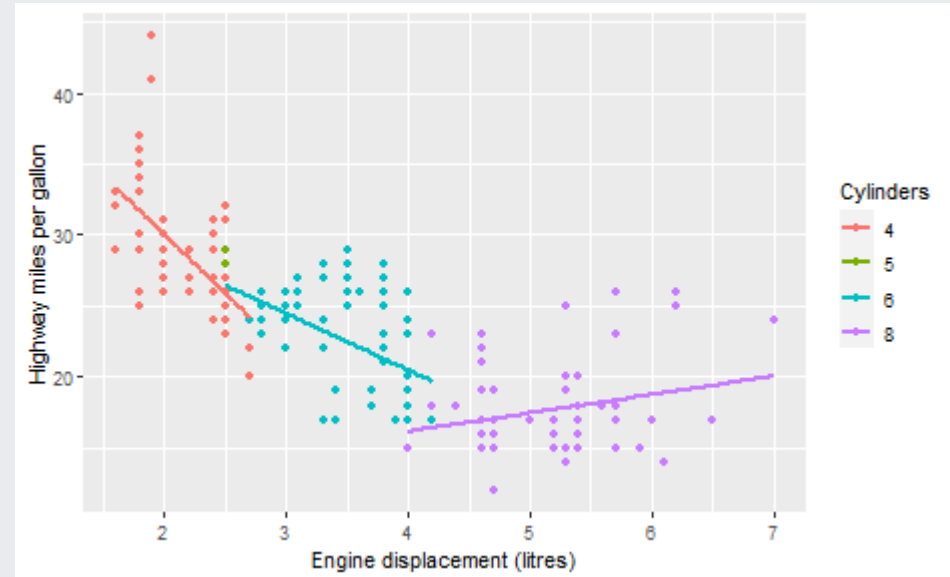


# Identifying groups



And we can also add linear regression lines for each grouping of cylinders, again using `geom_smooth()`.

```
ggplot(data = mpg,
       mapping = aes(x = displ,
                     y = hwy,
                     colour = factor(cyl)))
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(x = "Engine displacement (litres)",
       y = "Highway miles per gallon",
       colour = "Cylinders")
```



# Plotting categorical and continuous data

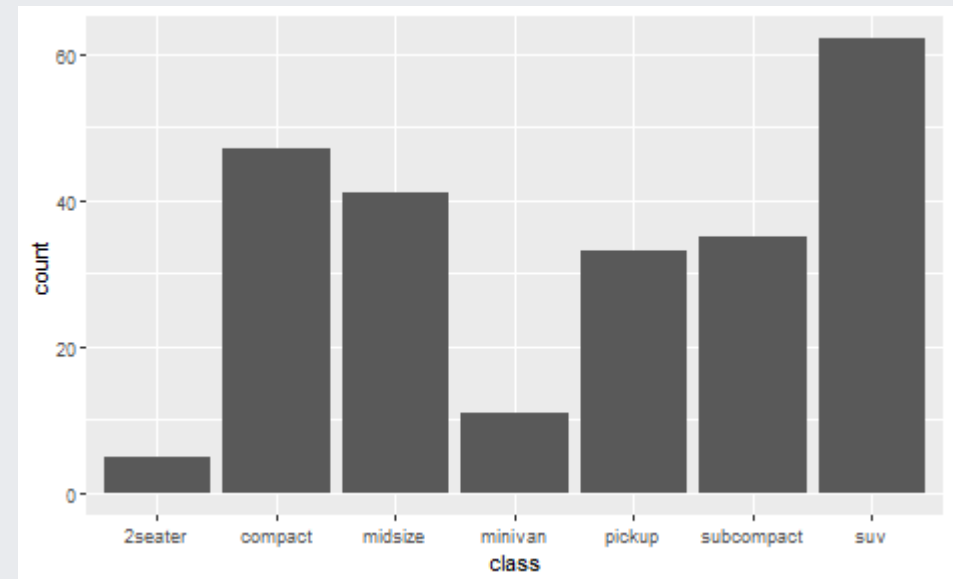
# Plotting a single categorical variable

Typically with a single categorical variable, we want a frequency count - i.e. we want to know how many times each category shows up.

A bar graph is ideal! For example, there are several different *classes* of vehicle in the *mpg* dataset. How many times does each one show up?

```
ggplot(mpg,  
       aes(x = class)) +  
  geom_bar()
```

`geom_bar()` will count for us, so we don't need to supply a `y aesthetic aes()`.



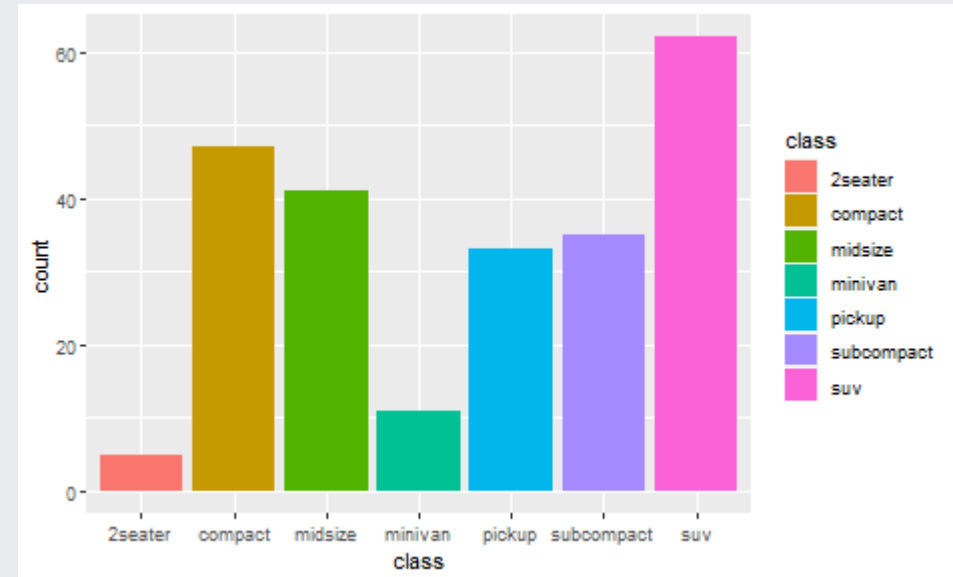
# Plotting a single categorical variable

As with plots we did earlier, the bars can be coloured in.

With `geom_point()` we change the colour aesthetic.

For `geom_bar()` we need to change the fill aesthetic.

```
ggplot(mpg, aes(x = class,  
                fill = class)) +  
  geom_bar()
```

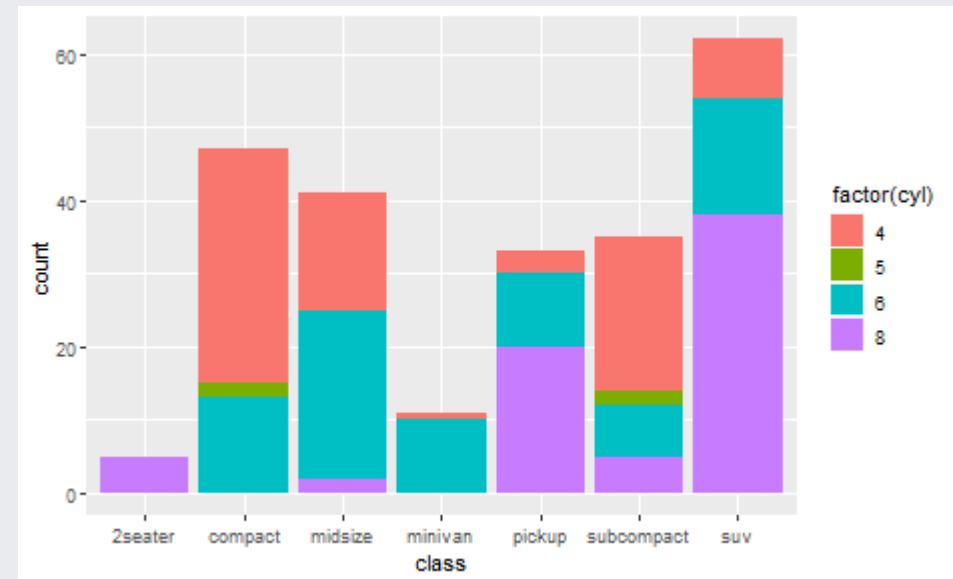


# Plotting multiple categorical variables

The fill doesn't have to use the same variable as the x variable.

For example, you may want to see how each count breaks down into groups of another categorical variable.

```
ggplot(mpg, aes(x = class,  
                fill = factor(cyl))) +  
  geom_bar()
```

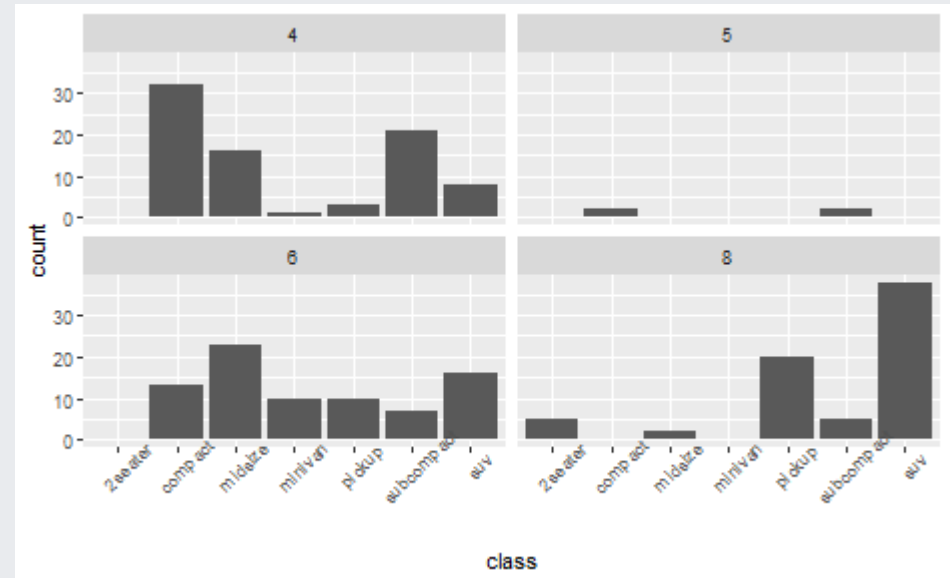


# Plotting multiple categorical variables

Alternatively, you may want to produce different graphs for each level of the other categorical variable

A nice way to do that is using **facets**, adding a `facet_wrap()` or `facet_grid()` layer to the *ggplot*.

```
ggplot(mpg, aes(x = class)) +  
  geom_bar() +  
  facet_wrap(~factor(cyl)) +  
  theme(axis.text.x = element_text(angle =
```



# Plotting continuous variables

# Plotting a single continuous variable

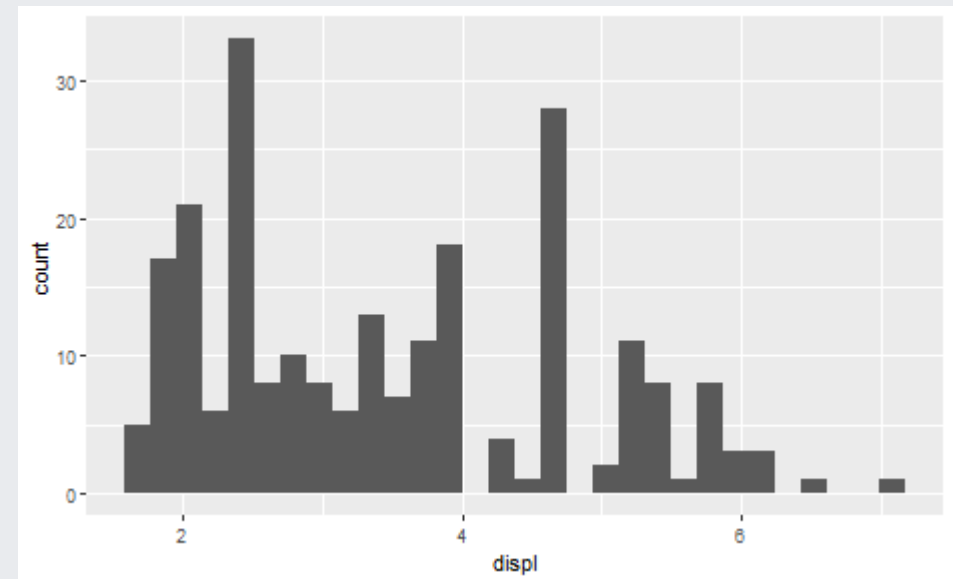
A lot of the time you'll be dealing with continuous, numerical variables.

What you often want to do is check how they are distributed (we'll go into this later in the course!).

Histograms split continuous variables up into discrete bins, and count how many of each value show up in each bin.

Here we use `geom_histogram()`. By default, it splits data into 30 bins.

```
ggplot(mpg, aes(x = displ)) +  
  geom_histogram()
```





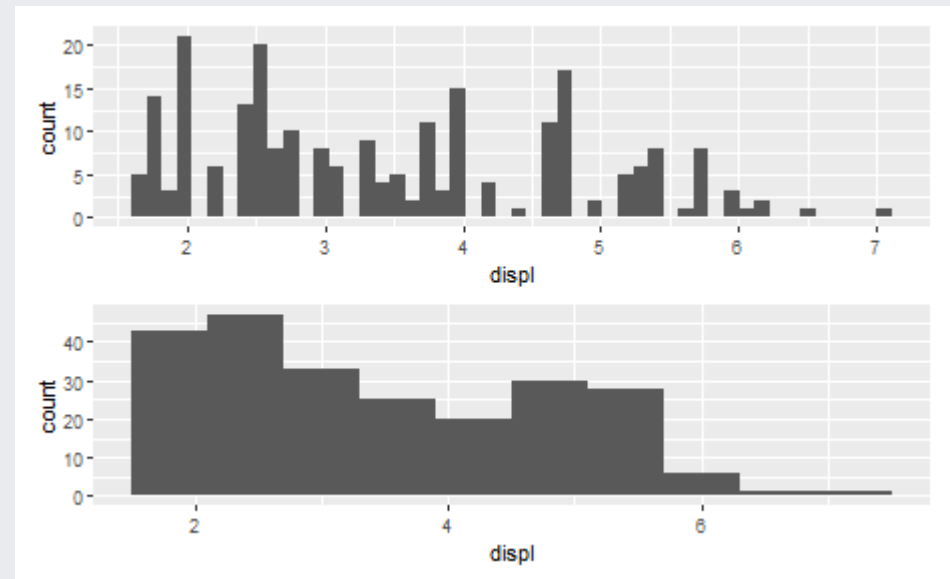
# Plotting a single continuous variable

Changing the number of bins can have quite dramatic results on the plots.

There are no hard and fast rules how many bins you need.

```
ggplot(mpg, aes(x = displ)) +  
  geom_histogram(bins = 50)
```

```
ggplot(mpg, aes(x = displ)) +  
  geom_histogram(bins = 10)
```



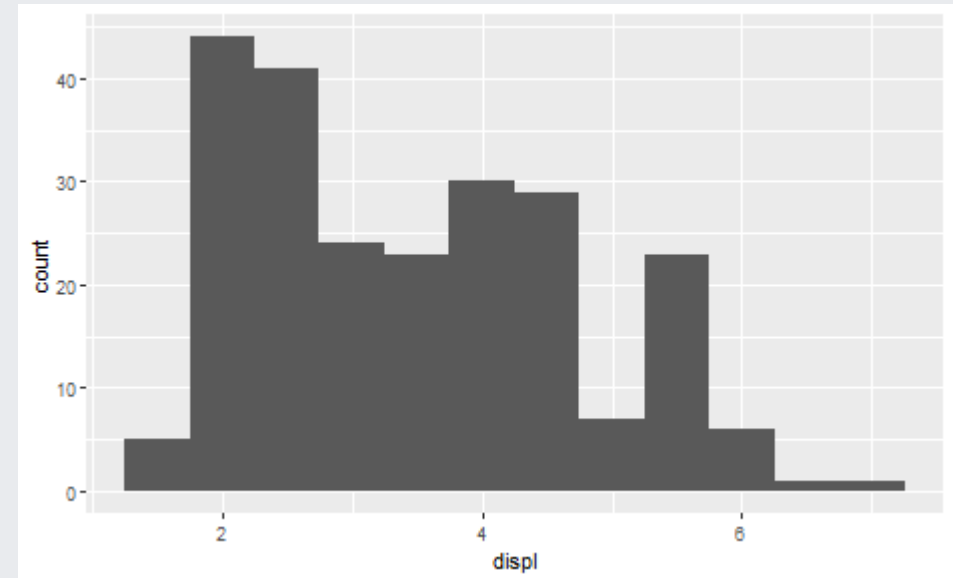
```
## Warning: package 'patchwork' was built under R version 4.2.2
```

# Plotting a single continuous variable

Rather than choosing a number of bins, you can also set the `binwidth`, in the same units as the variable.

For example, here it's set to make one bin every .5 units of the `displ` variable.

```
ggplot(mpg, aes(x = displ)) +  
  geom_histogram(binwidth = .5)
```



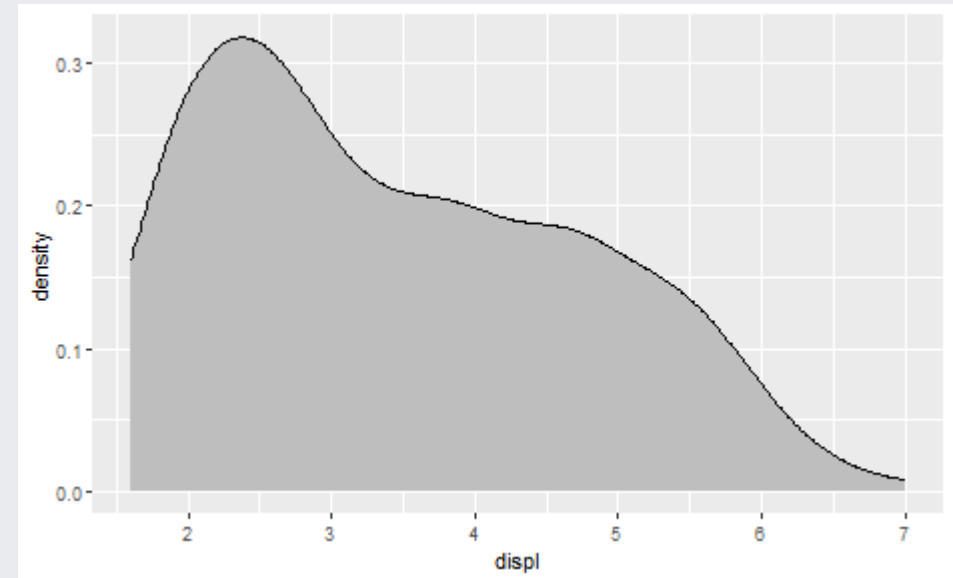
# Plotting a single continuous variable

An alternative to using a histogram is to plot a **kernel density estimate (KDE)**.

An advantage of the KDE (other than the fancy-sounding name) is that it provides smooth estimate over the range of the data and is much less dependent on an arbitrary parameter like "number of bins".

We draw a KDE using `geom_density()`.

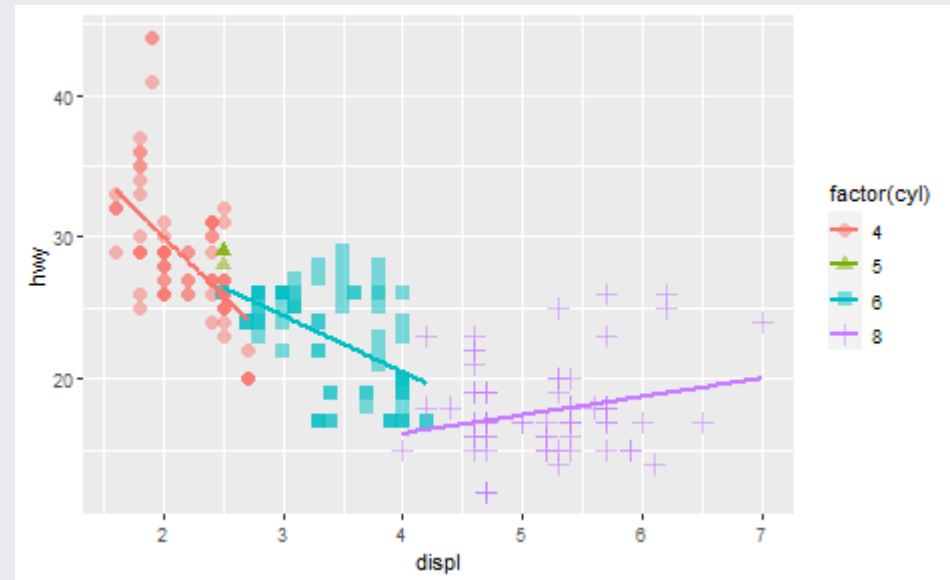
```
ggplot(mpg, aes(x = displ)) +  
  geom_density(fill = "grey")
```



# Plotting two continuous variables

The best type of plot for showing the relationship between two continuous variables is a **scatterplot**.

```
ggplot(data = mpg,
       mapping = aes(x = displ,
                     y = hwy,
                     colour = factor(cyl)))
  geom_point(size = 3,
             alpha = 0.5,
             aes(shape = factor(cyl))) +
  geom_smooth(method = "lm", se = FALSE)
```



**Continuous by categorical interactions**

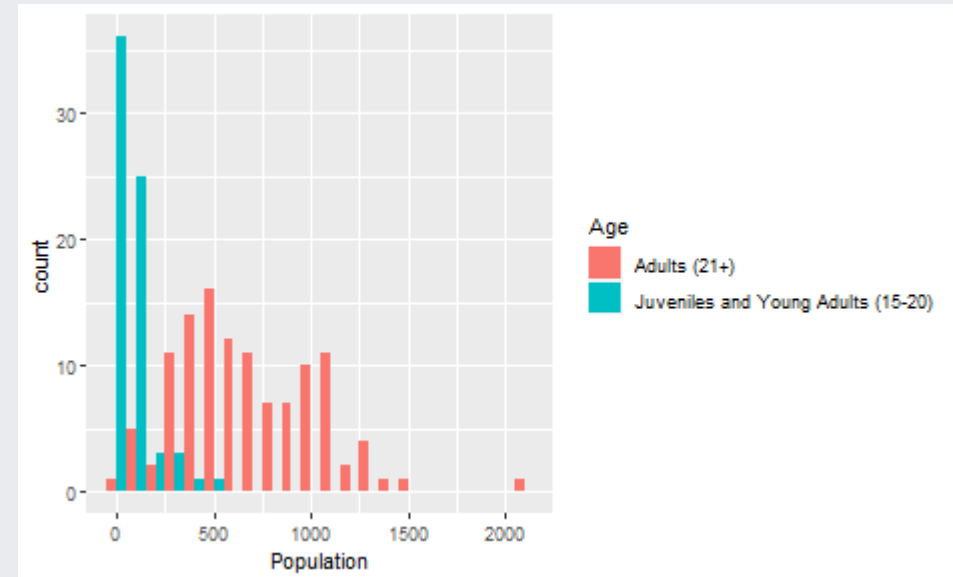
# Continuous by categorical interactions

Often when working with continuous data, you have additional categorical variables.

It's often easiest to put splits based on categorical variables side-by-side on the same plot.

Here we use `geom_histogram(position = "dodge")` to put the bars side-by-side.

```
ggplot(pris_pop,  
       aes(x = Population,  
           fill = Age)) +  
  geom_histogram(binwidth = 100,  
                position = "dodge")
```



# Continuous by categorical interactions

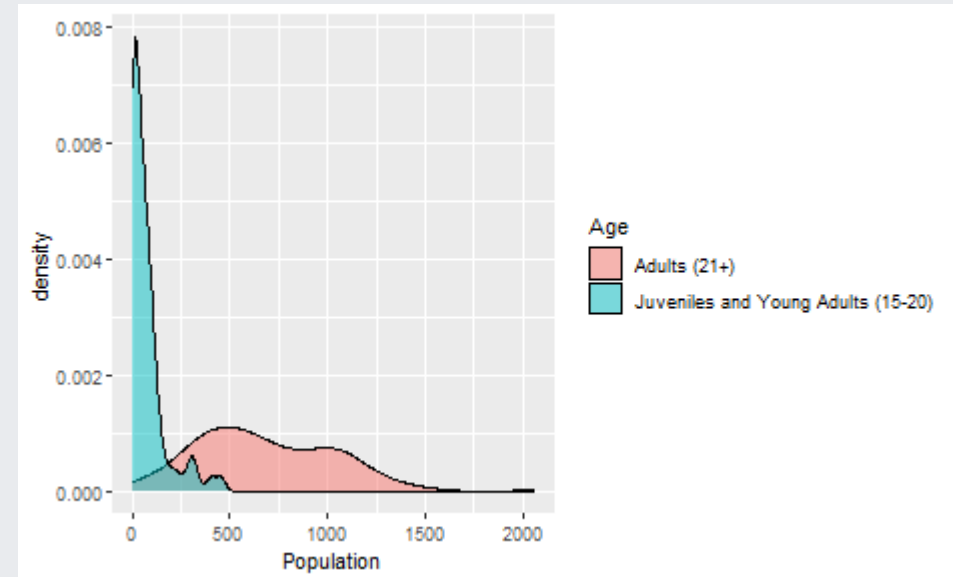
Another way to do this would be using kernel density estimates.

`geom_density()` uses the *fill* aesthetic for this.

Since the densities overlap, we can manipulate the *transparency* of the geom using the *alpha* argument.

Note that this can be applied to most *geoms* and is often useful when there is overlap.

```
ggplot(pris_pop,
       aes(x = Population,
           fill = Age)) +
  geom_density(alpha = 0.5)
```

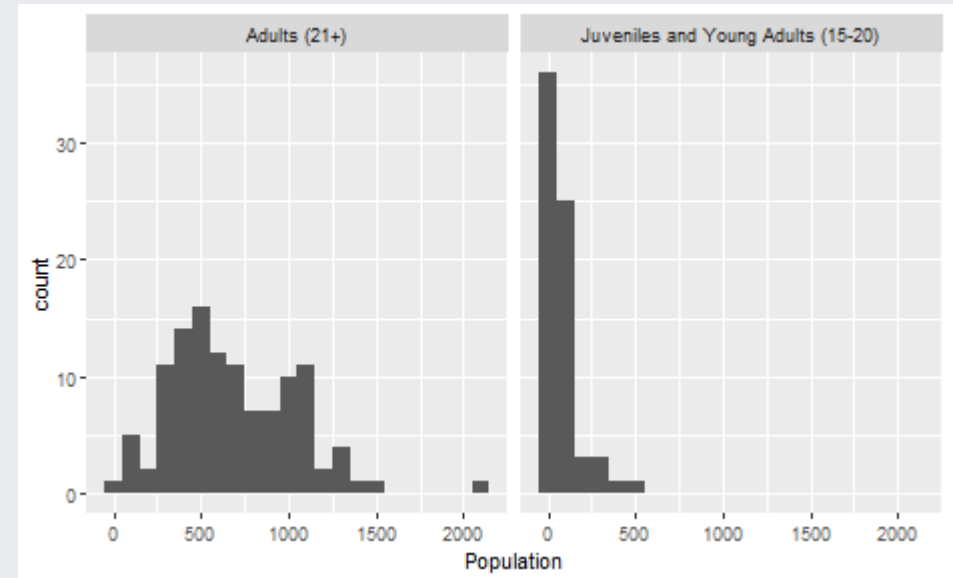


# Continuous by categorical interactions

However, sometimes you'll find it helpful to produce separate "panels" for each level of a categorical variable.

We can use the `facet_wrap()` or `facet_grid()` function to produce additional panels.

```
ggplot(pris_pop,
       aes(x = Population)) +
  geom_histogram(binwidth = 100) +
  facet_wrap(~Age)
```





# Continuous by categorical interactions

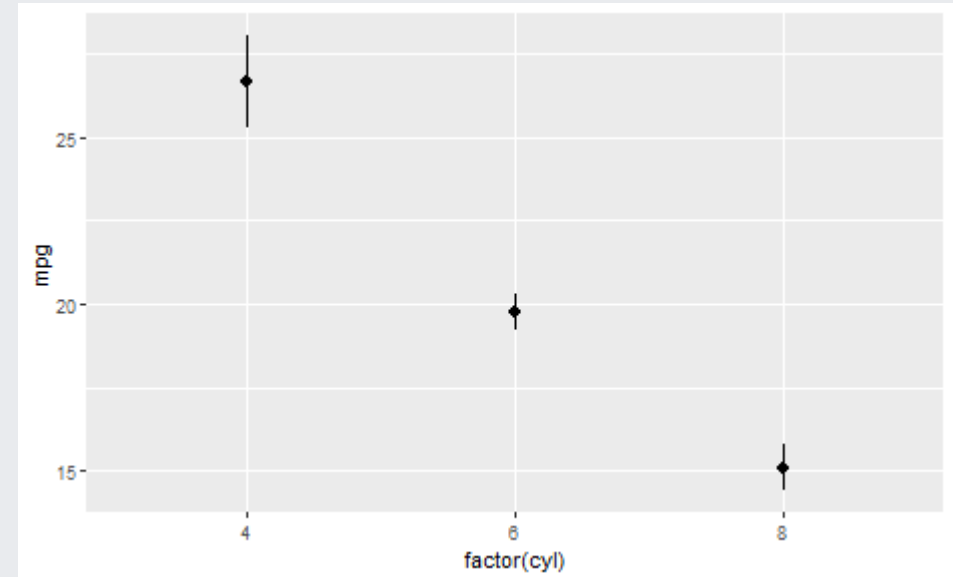
In the last few examples, we've plotted with the continuous variable on the x-axis.

We can also plot with a discrete variable on the x-axis.

In this case we want R to summarise the continuous variable, providing us with the mean and standard error for each level of *cyl* from the *mtcars* dataset.

We use `stat_summary()` to do this.

```
ggplot(mtcars,  
       aes(x = factor(cyl),  
           y = mpg)) +  
  stat_summary()
```

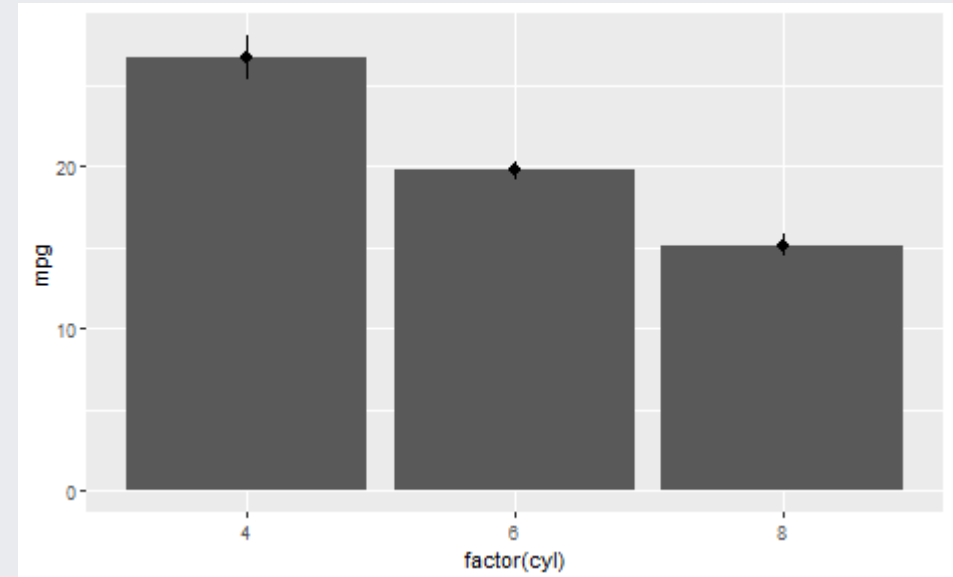


# Continuous by categorical interactions

Some people like to plot bar charts, with the mean and error bars overlaid on top.

We use `stat_summary()` twice, the first time specifying that we want bars using the *geom* argument, the second time just using the defaults.

```
ggplot(mtcars,
       aes(x = factor(cyl),
           y = mpg)) +
  stat_summary(fun = mean,
              geom = "bar") +
  stat_summary(fun.data = mean_se)
```



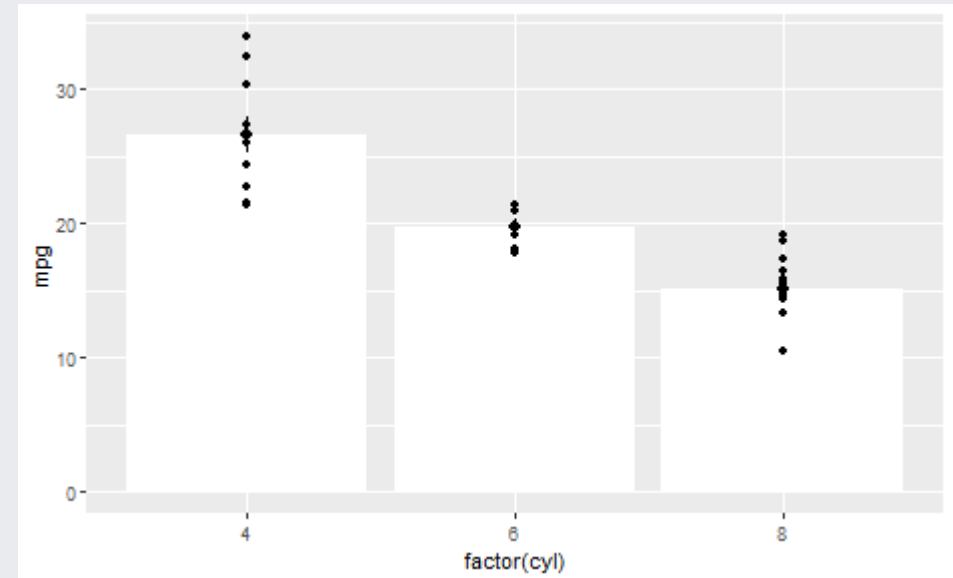
# Continuous by categorical interactions

But bar charts are not a very good way to show this kind of data!

Most of the space occupied by the bars has no data in it, as we can see when we add individual points with `geom_point()`.

Stick to using bars to show counts!

```
ggplot(mtcars,
       aes(x = factor(cyl),
           y = mpg)) +
  stat_summary(fun = mean,
              geom = "bar", fill = "white") +
  stat_summary(fun.data = mean_se) +
  geom_point()
```

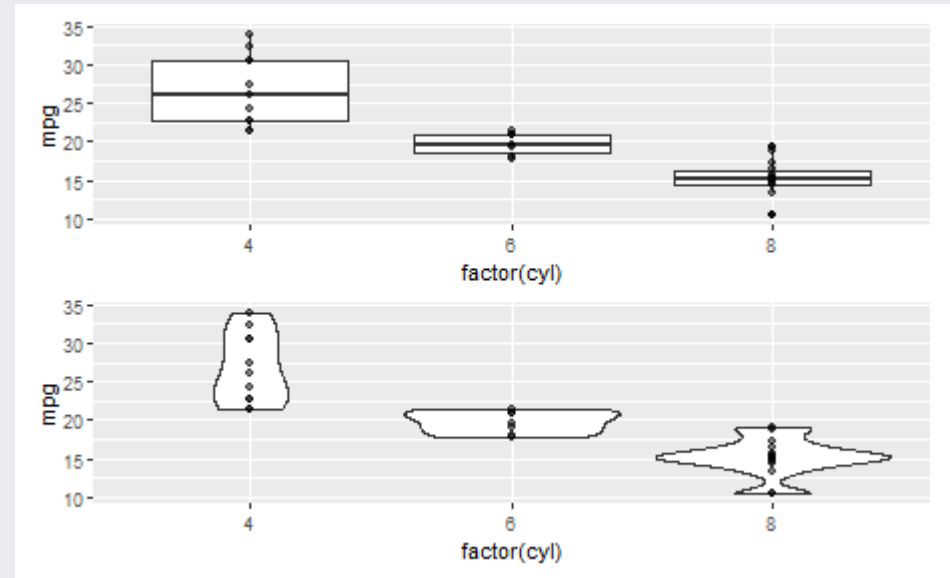


# Continuous by categorical interactions

Two better alternatives are **violin plots** or **boxplots**

```
ggplot(mtcars,
       aes(x = factor(cyl),
           y = mpg)) +
  geom_boxplot() +
  geom_point(alpha = 0.5)
```

```
ggplot(mtcars,
       aes(x = factor(cyl),
           y = mpg)) +
  geom_violin() +
  geom_point(alpha = 0.5)
```



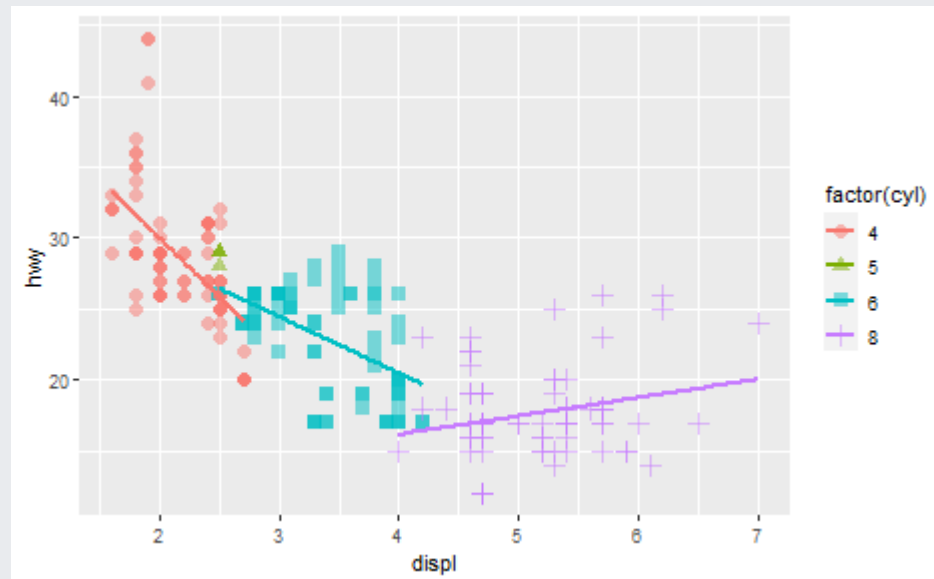
Jazzing up the plots

# Better labelling

Basic plot

Better labels

Fancier Plot



# Nicer overall theme

Themes

Basic plot

`theme_bw()`

BW plot

`theme_classic()`

Classic plot

**Themes** are the way `ggplot()` sets the overall look of the plots.

These can control things like:

- The colour of the background (e.g. grey or white)
- The presence of the gridlines in the background
- The choice and size of fonts for text

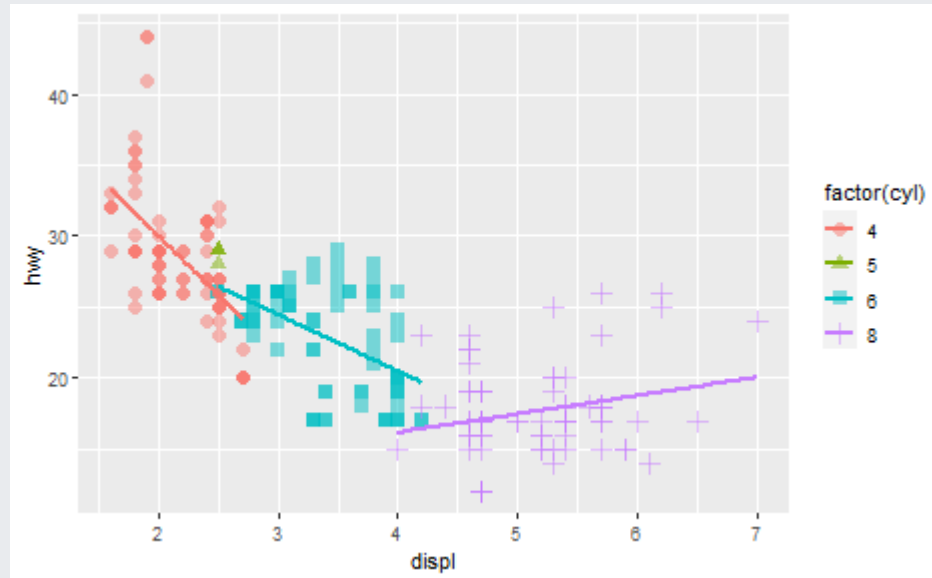
There are several default themes built in!

# Changing the colours

Basic plot

Colour brewer

New plot





# One final plot

Code

The plot

```
ggplot(data = mpg,
       mapping = aes(x = displ,
                     y = hwy,
                     colour = factor(cyl))) +
  geom_point(size = 3,
            alpha = 0.5,
            aes(shape = factor(cyl))) +
  geom_smooth(method = "lm", se = FALSE) +
  labs(x = "Engine displacement (litres)",
       y = "Highway miles per gallon",
       colour = "Cylinders",
       shape = "Cylinders",
       title = expression(~bold("Figure 1")),
       subtitle = expression(~italic("The relationship between MPG and Engine Displacement")) +
  scale_colour_brewer(palette = "Dark2") +
  theme_classic()
```

# Suggested reading

For practice of this week's concepts, see the RStudio.cloud [Visualize Data](#) primer.

For more general advice on plotting, see R4DS Chapters on [Graphics for Communication](#) and [Data Visualization](#), and Kieran Healy's [Data Visualization](#)

To prepare for next week, read R4DS Chapter on [Data transformation](#)