Online R Course

Contents

Chapter 1

Welcome!

In this tutorial we will be teaching R from the beginning - showing you how to set up R, building foundational R skills then moving onto how to plot and manage your data. Finally, we will introduce modelling in R.

We have written the tutorial in a **problem-orientated style**. Each chapter will introduce a problem and then we will use `R` to solve the problem. We have taken this approach as it should hopefully provide you with a context to approach your own problems with and, ideally, by the end of this tutorial, you will have learned enough to apply your skills to your own research questions and data.

We don't expect you to have memorised everything in these tutorials - it's unrealistic and it's not how data analysts work. Our main goal for the end of the tutorial is to teach you where to look and to be able to understand what you find so **you can learn R yourself** and when your your needs grow, you can solve the problems yourself.

The files for this tutorial are available for your personal use. If you reuse sections of this tutorial, please mention us and the appropriate citations found in the biblography.

If you have any comments or corrections, please contact us through https://stats4sd.org/contact

Chapter 2

Setting up

Unfortunately, as with any new skill, the first chapter can be the most text heavy. Apologies. We will do our best to;

- Keep it as brief as possible
- clearly structured so you can skip sections as needed
- complete so you know that after this chapter you will be focused on coding.

2.1 About R and RStudio

`R` is a progamming language that is primarily focused on statistics. This means that it's a way of talking to your computer to get it to solve certain problems and `R`'s speciality is statistics and data analysis.

It's not all it can do and is a great pathway to transition from point-and-click data analysis to more general programming. The skills you will learn along the way are part of most modern languages, so your time will be well spent!

There are many reasons individuals use `R` but for our needs the main selling points are;

- Integrated platform for all data management, analysis and graphics
- Allows for reproducible research
- Active community of users providing support
- Cost = Free!

2.1.1 Installing R and RStudio

`R` comes bundled with an interface but not many people use it. The vast majority of `R` users actually use **RStudio** to work with R including us. RStudio provides a lot of useful features such as making it easier to see what you've done so far and view the help menu.

There are two options at the time of writing; installing `R` and RStudio on your personal computer (local) or using an online version (cloud). Either is fine. The cloud version is, at the time of writing, free but this is likely to change. Also the cloud version can be slow. Basically, if you are uncomfortable or unable to install programs on your computer, then use the cloud version. Otherwise, the local version is the best choice.

2.1.1.1 Local

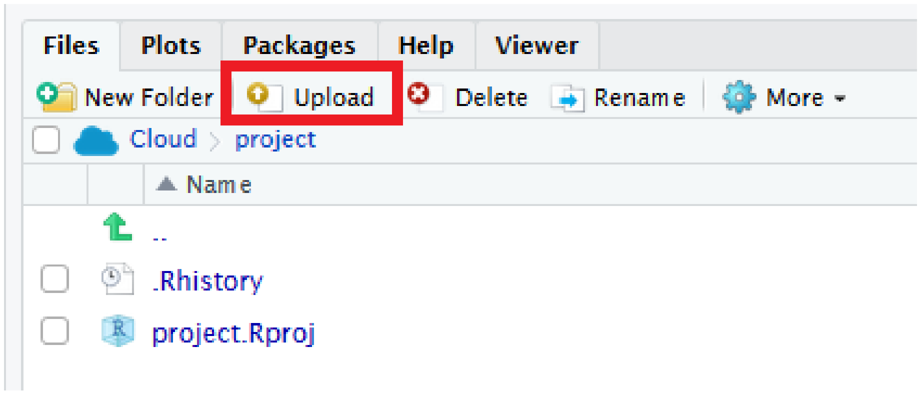
We recommend following the "Download, Install and Setup R and RStudio" tutorial on https://www.statslectures.com/r-stats-videos-tutorials/getting-started-with-r

The direct link is here; https://youtu.be/riONFzJdXcs

2.1.1.2 Cloud

As of the time of writing, RStudio is providing a free version of their software that you can run in your web-browser. Go to https://rstudio.cloud and sign up as you would most online accounts.

One thing to note is getting data onto the cloud requires an extra step. You'll need to press `Upload` in the bottom right panel, shown below, and browse your computer for the files. Also, if you want to upload multiple files at once, you need to first "zip" these files together and upload this file.



2.2 Get the data

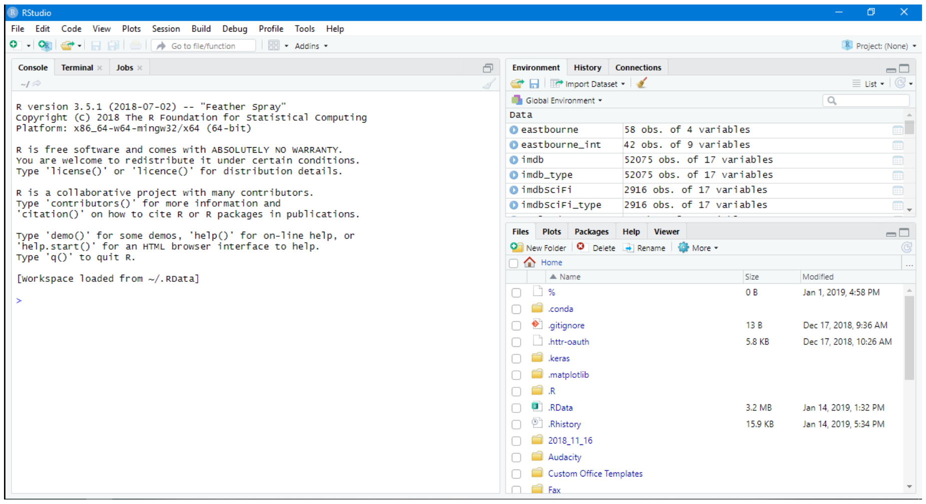
The data sets can be currently found at the following link.

In case the link is broken, the updated link is likely to be found by navigating to https://stats4sd.org/resources, searching "Online R Course" and downloading `Data.zip`.

2.3 Navigating RStudio

Open RStudio by either opening the program on your computer as you normally would or by going to https://rstudio.cloud, logging in and then pressing new project.

You will be faced with something like this;



```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

>
```

```
Data
eastbourne        58 obs. of 4 variables
eastbourne_int    42 obs. of 9 variables
imdb              52075 obs. of 17 variables
imdb_type         52075 obs. of 17 variables
imdbSciFi         2916 obs. of 17 variables
imdbSciFi_type    2916 obs. of 17 variables
```

Name	Size	Modified
%	0 B	Jan 1, 2019, 4:58 PM
.conda		
.gitignore	13 B	Dec 17, 2018, 9:36 AM
.httr-oauth	5.8 KB	Dec 17, 2018, 10:26 AM
.keras		
.matplotlib		
.R		
.RData	3.2 MB	Jan 14, 2019, 1:32 PM
.Rhistory	15.9 KB	Jan 14, 2019, 5:34 PM
2018_11_16		
Audacity		
Custom Office Templates		
Fax		

If you're overwhelmed by the options, don't worry! We will show where the most important sections.

The first thing to notice is there are three main sections. Each of these sections has "tabs" in the same way your web browser does.

2.3.1 Console

The left most and largest section main tab is called "Console". This is where you type you R code. For example, type 2+2 and then press Enter. You've just run your R command.

2.3.2 Environment

To the top right there is a single important tab; "Environment" which tracks all the things you've made during your current working time with R.

2.3.3 Packages, Help

The bottom right panel has two panels we will use; packages and help. Both will be covered later but briefly packages are where R extensions are listed and Help is where we can access support documents about R in a convenient way.

2.3.4 Source

We are about to introduce our fourth and final panel. Above the "Console" there is a button called `File`. Do the following; `File -> New File -> R Markdown`. It might take a little while to load the first time as it is setting up, just be patient. We will talk about R Markdown in the next session but a quick note; this new section is generally called "Source" because this is the source of the R Code you have written which is then sent to your console.

If you look at the bottom of your window, you should see "Console", which was minimised when you opened Source. As you with any other program, to the right you can use the split-screen and maximise button.

To recap, the main tabs you need to know are "Console", "Environment", "Help", "Source", and "Packages"

QUESTION: Can you open each of these panels?

2.4 R Markdown

R Markdown are documents that allow you to write text around your code and output your work into multipile formats, including a HTML, PDF or Word.

The "text mode" is similar to your experience with typing documents in Word or Notepad. Technically it uses "Markdown", so you have the option for tables and numbered lists in your final output.

Code in R Markdown is implemented in "chunks". There are two types of chunks, in-line and blocks. To insert a chunk, you need to press `Insert -> R` located at the top right of the "Source" panel.

Once done, enter the code below and press the green play button on the right hand side of the chunk. In this tutorial, code appears in a light grey box and output in a white box.

Code

```r
2+2
```

Output

```
## [1] 4
```

Hopefully when you run the code, you see that 2+2 still equals 4!

The code chunks in R Markdown start with ` ```{r} ` and ends with ` ``` `.

Do not modify these lines of the chunk otherwise bad things might happen. Modify anything in the middle.

2.5 Comments

Using `#` in a code block stops the anything after the symbol being run. `#` is referred to as "commenting" code because it's usually used to add commentry to long pieces of code. Try running the code below.

QUESTION: What do you think the code below will output? Why?

```r
#I am trying to calculate the following;
2 + 2 # + 2
# + two
```

2.6 Getting the files

For this tutorial you will need to download a collection of files. These can be found at here. One you have downloaded them, you will need to extract them.

Chapter 3

Base-R

Congratulations! By this point you should have successfully installed R and R Studio. If not, see the previous chapter.

Specifically you have installed "Base-R". R is an extendable language, which means you can expand it's vocabulary to solve problems in new and simplier ways. It's why it's so fresh and on the cuting edge.

In this tutorial, we're going to start with the standard language; "Base-R". It also provides the foundation for later topics. "Base-R" is by no means basic! There are many people who only use base-r for the entirety of their projects. It's **Base**-R not **Basic**-R.

3.1 Objects and functions

Within R there are two fundemental concepts; objects and functions. Objects are a way of holding information. Functions are a way of manipulating objects. This may seem really complicated but the code below demonstrates this concept.

```
2+2
```

```
## [1] 4
```

QUESTION: In this example there are 2 objects and 1 function. Can you name them?

.

.

.

The objects are `2, 2` and the function is `+`.

To create custom objects in R we use `<-`, pronounced “assign”.

```
x <- 2+2
```

This would read as “assign the object `x` the value of 2+2”.

What do you see underneath the chunk this time? Does 2+2 no longer = 4?

When objects are assigned they usually do not return any output directly. Check over in the Environment pane of RStudio on the right hand side of the screen?

Can you see something called x? What does the environment say about x?

You could also print this out more explicitly by submitting the name of the object in an R chunk.

```
x
```

```
## [1] 4
```

If you skipped the previous chunk (because it seemed obvious - you know what 2+2 is!) you might have got an error. You need to create an object x before you can look at it. If you missed running a previous chunk, or like skipping to the end then you can press the button to the left of the play button “run all chunks above” (grey triangle above a green line)

3.1.1 Naming objects

When giving things names in R it is better to be a bit more informative than using single letters (x,y,z etc.). We can give R objects any name we want to - but there are a few rules: 1. No punctuation except _ and . 2. No spaces 3. Only standard english alphanumeric characters - no accents 4. Names can include numbers but can’t start with numbers

This is valid

```
sjkfjhskjdhsajsfgldsjghajfhljhgsdlk <- 2+2
```

But really stupid - we want our names to be short, clear and memorable

3.1.2 Errors

R is also case sensitive - try running this:

```
X
```

```
## Error in eval(expr, envir, enclos): object 'X' not found
```

Get used to that error! We have an object called `x` but we don't have anything called `X`. Capitalisation and spelling is vital.

As a new user remember that about most of your initial errors are likely to be found by checking the **B.S.Q.C. (brackets; spelling; quotation marks; case)**, or the result of problems with sequencing of loading data or packages.

Its also a good idea to avoid names which are used elsewhere in R for functions as this can cause problems with duplication and/or confusion.

R is big so there are lots of names used for things, so sometimes it happens but try to avoid as much as possible.

Speaking of functions...

3.2 Functions

A function is something that takes in a thing (input) and returns a thing (output). Nearly all functions get called like this

`functionname(input)`

Most functions also work like this `functionname(input_1, input_2, ...)`

A super useful function is c() and is everywhere is R, but does not have a very informative name. It is short for **c**ombine since it **c**ombines a bunch of stuff together.

```
c(1,2,3,4,5)
```

```
## [1] 1 2 3 4 5
```

Think of `c()` taking the inputs and making a list from them. In our example, the first item in the list is `1`, the second being `2`, ect.

Question: Modify the chunk above so that this is assigned to an object called y

You could now use another function to get ther average of those numbers

```
mean(y)
```

```
## Error in mean(y): object 'y' not found
```

Remember - this line will only work if you succesfully assigned the object y.

```
y<-c(1,2,3,4,5)
mean(y)
```

```
## [1] 3
```

3.2.1 Help

Throughout this tutorial you will be introduced to a lot of new functions with even more options. **We don't expect you to memorise them**. You are learning **where** to look.

You can get help within R by using a question mark followed by the name of the function.

```
?mean
```

The help files will list the options and a brief description. Once you know what you are looking for, it's easy to read.

There are always worked examples in the help file along with each function, which are often more useful than the help menus themselves. This can be found using `example()`

```
example(mean)
```

```
##
## mean> x <- c(0:10, 50)
##
## mean> xm <- mean(x)
##
## mean> c(xm, mean(x, trim = 0.10))
## [1] 8.75 5.50
```

But `?` and `example()` only work if you know the name of the function!

There are other ways of checking within R for how to do things; but at this point you are much better off heading to Google.

https://www.google.co.uk/search?q=How+do+I+calculate+a+mean+in+R

3.3 Packages

Up to this point, we've been using Base-R.

A really nice thing about R is that anyone can write extensions to R called packages. There are lots of really useful packages that make working with R much easier, others that let you make really nice graphics, or fit clever statistical models.

In this course we will use some of these packages so let's set up now by making sure we have everything we need. To get them we need to download them from the internet and install them into our version of R. To do so, run the following code;

```
install.packages(c("ggplot2","dplyr","openxlsx", "tidyr", "plotly"))
```

We will talk about these packages when we need them later - don't worry too much about what each of them do for now.

3.4 Quotation Marks

Also note that each of the packages in the previous line are in quotation marks. This is because they are not things that exist in our current session of R (yet).

Look at the difference between these two lines

```
x
```

```
##  [1]  0  1  2  3  4  5  6  7  8  9 10 50
```

```
"x"
```

```
## [1] "x"
```

Something in quotation marks is treated literally as what is there. Something not in quotation marked is evaluated.

If we use the name of something not in quotation marks and it is not the name of something already in our R session we get an error

Chapter 4

Graphics with ggplot2

4.1 Welcome to ggplot

Graphs play an important part of any data analysis in understanding your data.

As we've mentioned in the previous chapter, `base-R` can be extended using packages and the particular package we'll be using in this tutorial will be `ggplot2` (Wickham et al., 2019a).

There are lots of of different systems for producing graphs in R. We are going to focus on the `ggplot2` package. The syntax can look scary at first, but once you understand the basic building blocks, you will be able to produce complex graphs very easily.

Comparing `base-R` with `ggplot2`

baseR

- Simple intuitive syntax for basic plot
- Pretty ugly basic plot
- Inconsistent syntax for different plot types
- Complicated syntax for customisation

ggplot2

- Unusual syntax for basic plot

- Pretty looking basic plot
- Consistent syntax for different plot types
- Consistent syntax for customisation

1 against 3. ggplot2 wins.

4.2 Setting up

Assuming you have followed the previous chapter, particularly from 2.1.1, at this stage you have installed `R`, RStudio, familarised yourself with the key panels and then installed key packages including `ggplot2`.

This means you simply need to load `ggplot2` into your current session by simply running the command below.

```r
library(ggplot2)
```

To plot our data, we first need data. Luckily we have some prepared for you.

The simpliest way for this tutorial is to look at the `Environment` panel in the top right of RStudio and press the folder with the green arrow. Navigate to the Data folder you downloaded from GitHub (Link Here). Double click on `03 Pulse.RData`. In a later chapter, we will talk about how to import other formats.

4.3 Description of the "Pulse of the Nation" Dataset

Before we proceed with any analysis, and even outside this tutorial, it is important to understand your dataset.

You can view the dataset in RStudio by single clicking the name in the Environment tab.

The "Pulse" dataset is an extract from the monthly survey "Pulse of the Nation" made by Cards Against Humanity (Cards Against Humanity, 2018) which contains a representative sample of US citizens.The dataset contains an extract of 356 repsonses to 10 of the survey questions:

Variable	**Question**
Gender	What gender do you identify with?

Variable	Question
Age	What is your age?
AgeGrp	Age range
Race	What is your race?
Income	About how much money do you make per year? (USD)
Education	What is your highest level of education?
PoliticalParty	In politics today do you consider yourself a Democrat a Republican or Independent?
PoliticalView	Would you say you are liberal conservative or moderate?
ApproveTrump	Do you approve disapprove or neither approve nor disapprove of how Donald Trump is handling his job as president?
Attractiveness	On a scale of 1-10 how physically attractive are you?

4.4 Main components of ggplot2

To build a plot in `ggplot2`, you have to specify features of the plot. The two essential ones are;

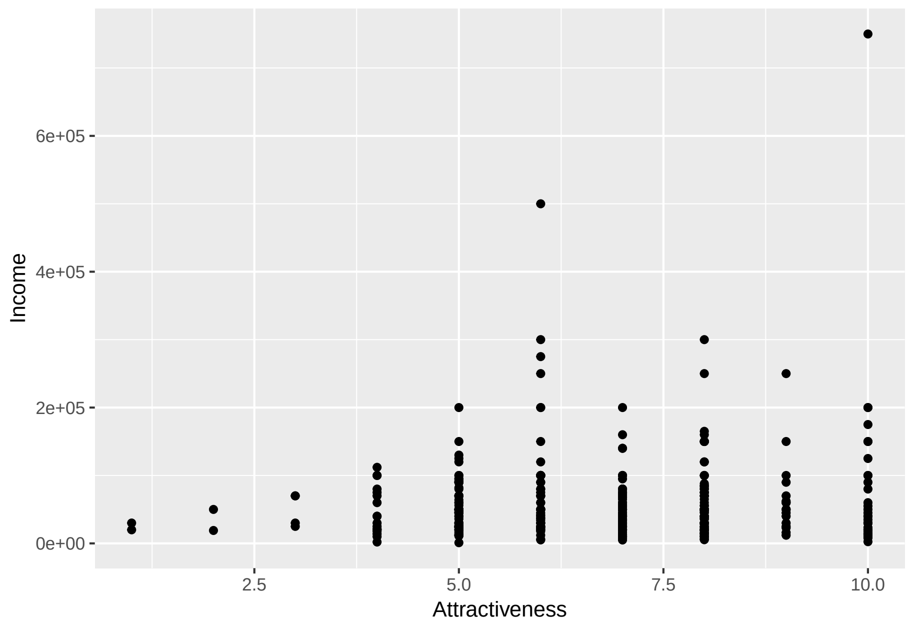
- Aesthetics - `aes()` - describes which variables in our dataset are going to be mapped to what components of our plot
- Geometries - `geom_****()` - describes how the variables will be plotted

The other types of features are; Scales, Themes, Facets which will be introduced throughout this tutorial.

4.5 First ggplot2 graph

Here is how you produce a scatter plot of perceived attractiveness against age

```
ggplot(data = Pulse, aes(x = Attractiveness, y = Income)) +
  geom_point()
```



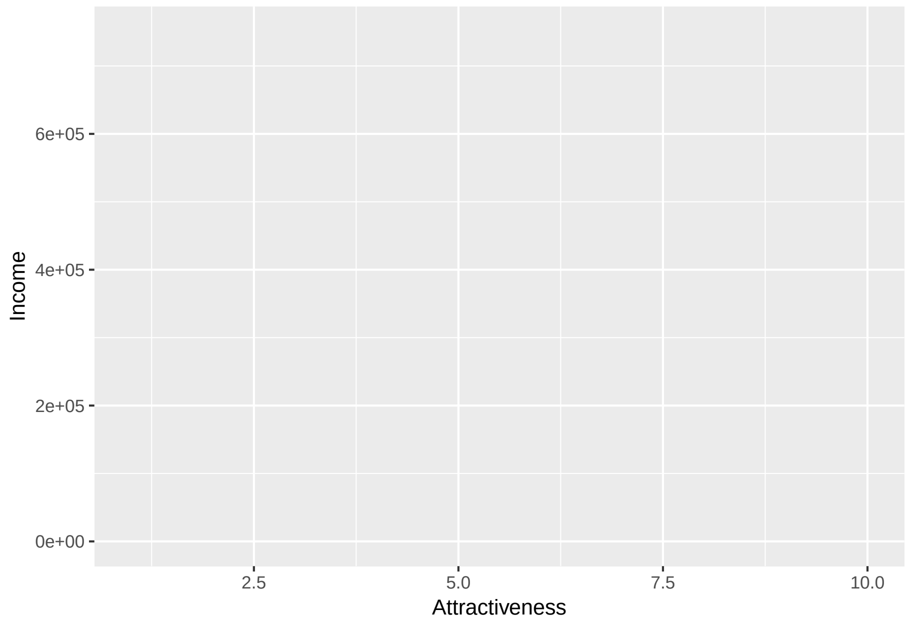
Before we break down this example, it is perhaps worth noting the notation used on the y axis. It is how computers show scientific notation where `2e+05` $= 2 \times 10^5$.

Now let's look at the different parts of the commands.

With `ggplot2`, you always begin a graph with the function `ggplot()`, which creates the base layer and sets the default for each plot but it is blank until we add some layers - "geoms" - to it.

The `aes` argument defines which variables you want to use for which purpose. '`aes`' is short for aesthetics. So `ggplot(data = Pulse, aes(x = Attractiveness, y = Income))` defines the data and the variables to be plotted on the x-axis and the y-axis.

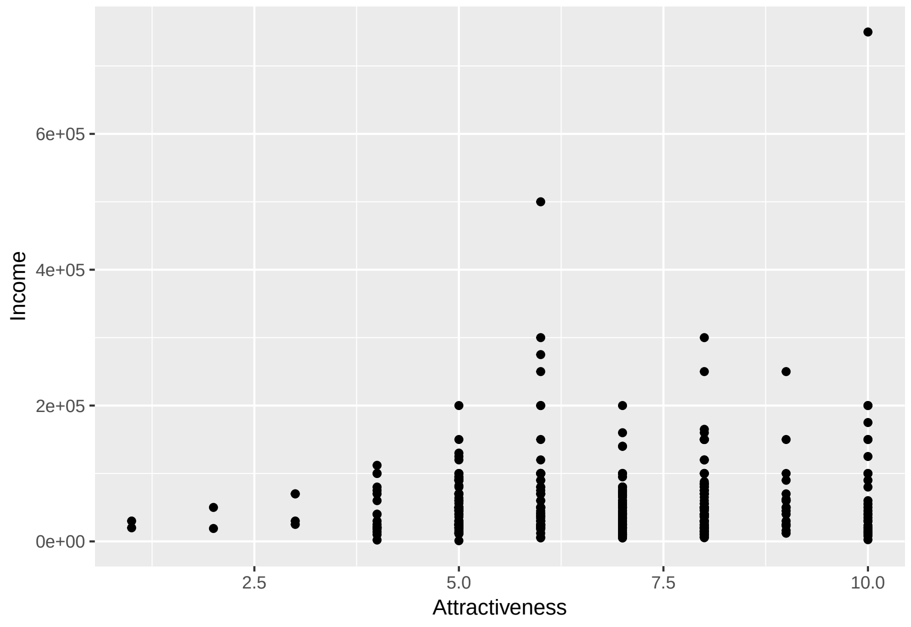
```
ggplot(data = Pulse,  aes(x = Attractiveness, y = Income))
```



As you can see - not a very informative graph!

You complete your graph by adding one or more layers to `ggplot()`. There are many different types of layers you can use to create your graph and they all start with `geom_????`. For example, the function `geom_line()` adds a layer of a line to your plot.

```
ggplot(data = Pulse,  aes(x = Attractiveness, y = Income)) +
  geom_point()
```

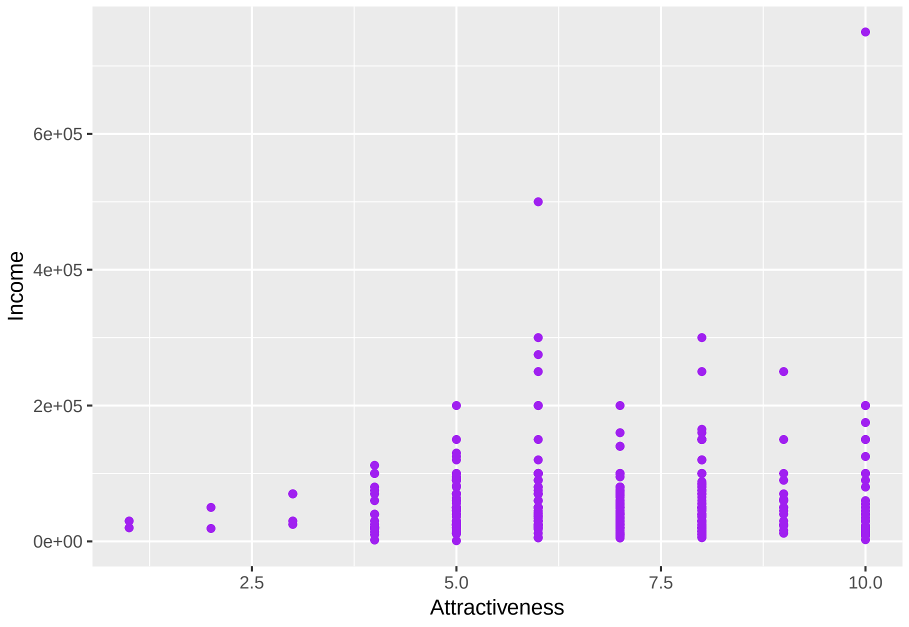


QUESTION: Modify the same command to instead plot a scatter plot of age on the x axis against income on the y axis

4.6 Adding colour

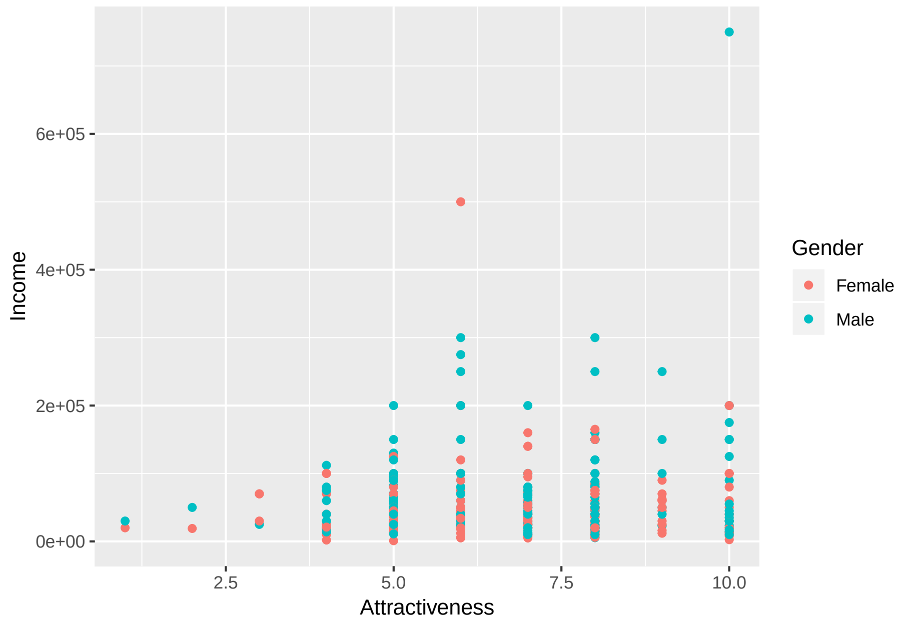
We can modify the appearance of our points in one of 2 ways - either we can map a variable of our data to an aesthetic or we can permanently apply a style characteristic across all points. To make all the points purple we can add colour="purple" into the geom_point() part of the syntax.

```r
ggplot(data = Pulse,  aes(x = Attractiveness, y = Income)) +
  geom_point(colour="purple")
```



To set the points to be different colours based on the column ‘Gender’ then we can map gender to the colour aesthetic:

```r
ggplot(data = Pulse,  aes(x = Attractiveness, y = Income, colour=Gender)) +
  geom_point()
```



QUESTION: Before pressing run - predict what will happen in the following two chunks of code. After running - can you explain why?

A

```
ggplot(data = Pulse,  aes(x = Attractiveness, y = Income, colour="purple")) +
  geom_point()
```

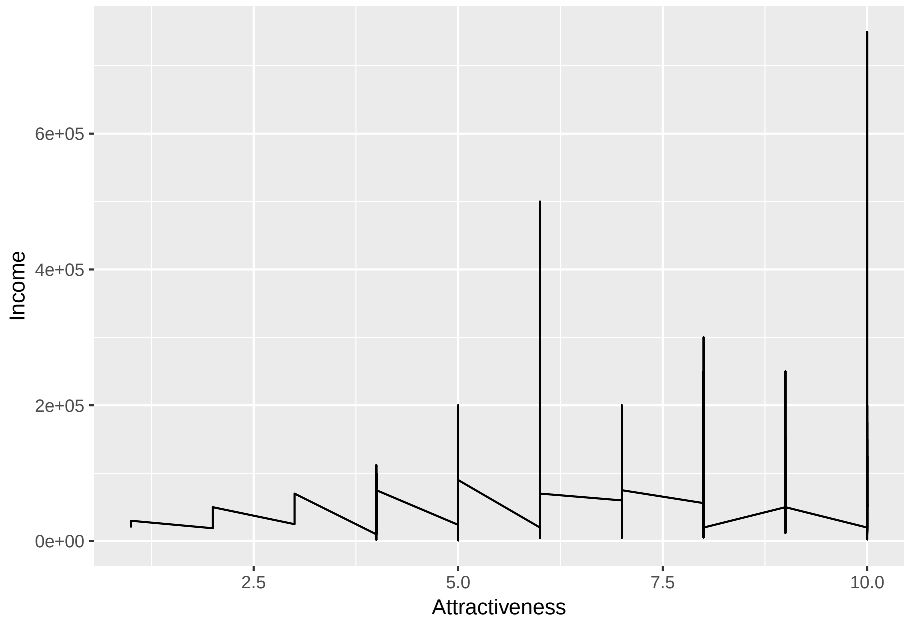
B

```
ggplot(data = Pulse,  aes(x = Attractiveness, y = Income)) +
  geom_point(colour=Gender)
```

4.7 More Geoms

Let's go back to attractiveness. If instead we wanted a line plot we can simply replace `geom_point()` with `geom_line()`.

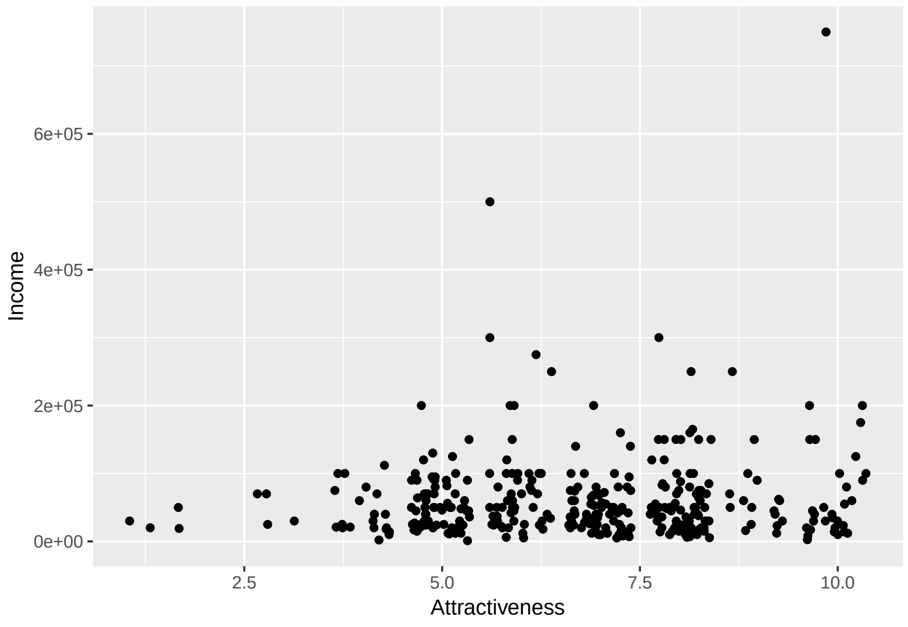
```
ggplot(data = Pulse,  aes(x = Attractiveness, y = Income)) +
  geom_line()
```



This graph is not particularly helpful in conveying any useful information about our data. Just because something is possible, does not mean it is advisable.

A geom that may be more useful is `geom_jitter()` - which would help us here because the attractiveness variable has a restricted set of values, so many points are being overlayed

```r
ggplot(data = Pulse,  aes(x = Attractiveness, y = Income)) +
  geom_jitter()
```



The `geom_****` functions all have various default options associated with them. For example, the size and colour of the points in `geom_point()` or the amount of jitter associated with them in `geom_jitter()`.

QUESTION: Using the help menu for geom_jitter can you modify the following code so that the amount of horizontal jitter is reduced and that there is no vertical jitter applied

```
?geom_jitter
```

```
ggplot(data = Pulse,  aes(x = Attractiveness, y = Income)) +
  geom_jitter()
```

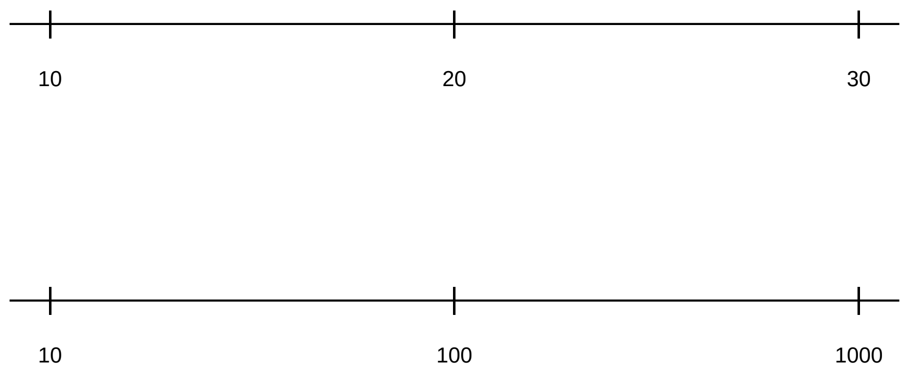
4.8 Scales

Income is a notoriously fickle variable. As we can see from the the table below there is a huge range in the values income takes, so when we have plotted Income against Attractiveness, most of our points are bunched up near the bottom.

We can show this by categorising our points in a table such as below;

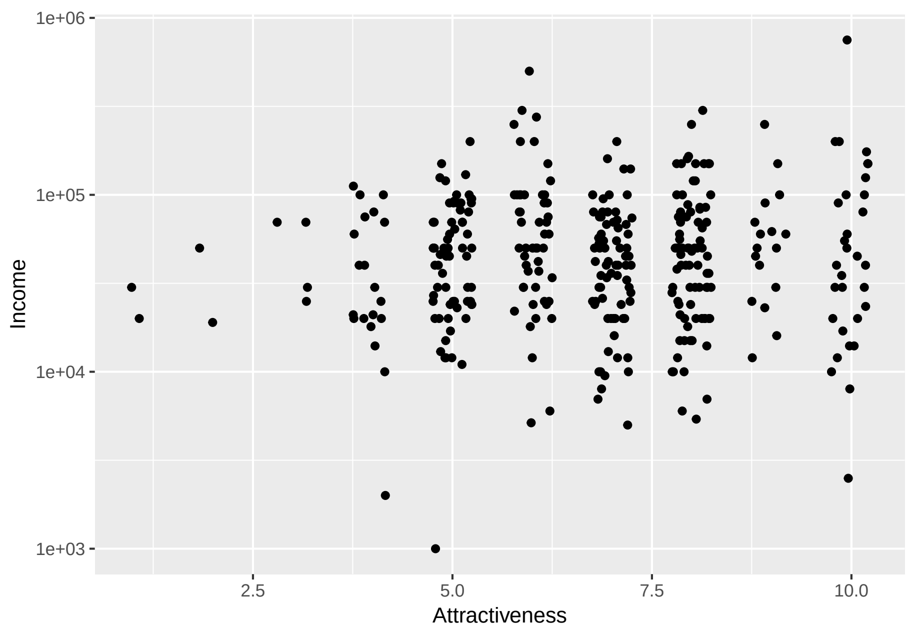
Income	Freq
$1,000-$9,999	13
$10,000-$99,999	278
$100,000-$999,999	59

One solution is to apply a log scale for income. A log scale is when each tick mark of the graph represents an increase in the power.



In the above example, the top line is increasing *by* 10 whereas hde bottom line is increasing *by powers of 10*. To apply this change of scale to our plot, we add the following code to our plot. `scale_y_log10()`

```
ggplot(data = Pulse,  aes(x = Attractiveness, y = Income)) +
  geom_jitter(width=0.25,height=0)+
    scale_y_log10()
```

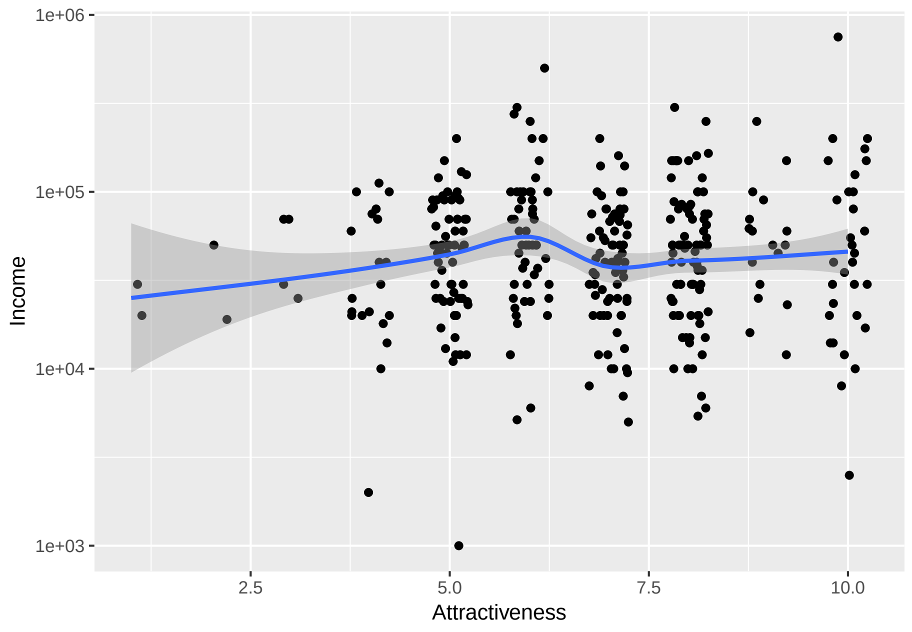


4.9 Layering

ggplot2 graphs can have more than one layer. We may want to produce a line plot with the points shown on top. In ggplots we can simply add more layers to a graph by using `+`. Layers are added sequentially so the final layer in the code will be plotted over the existing layers. We might want to add a trend line onto our plot to show the increasing trend of income with attractiveness (or the increasing arrogance of wealth?)

```
ggplot(data = Pulse,  aes(x = Attractiveness, y = Income)) +
  geom_jitter(width=0.25,height=0)+
    scale_y_log10()+
      geom_smooth()
```

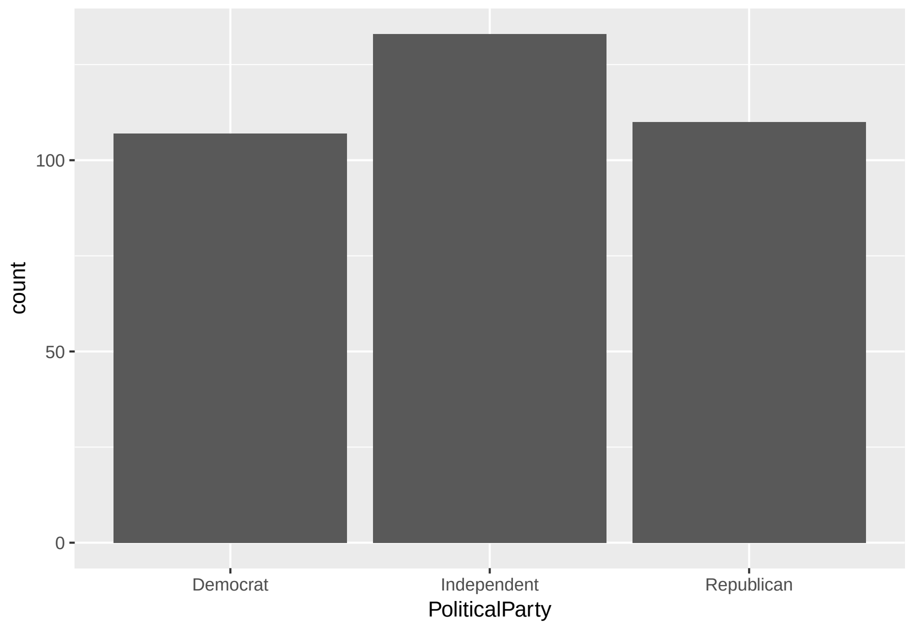
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



4.10 More geoms

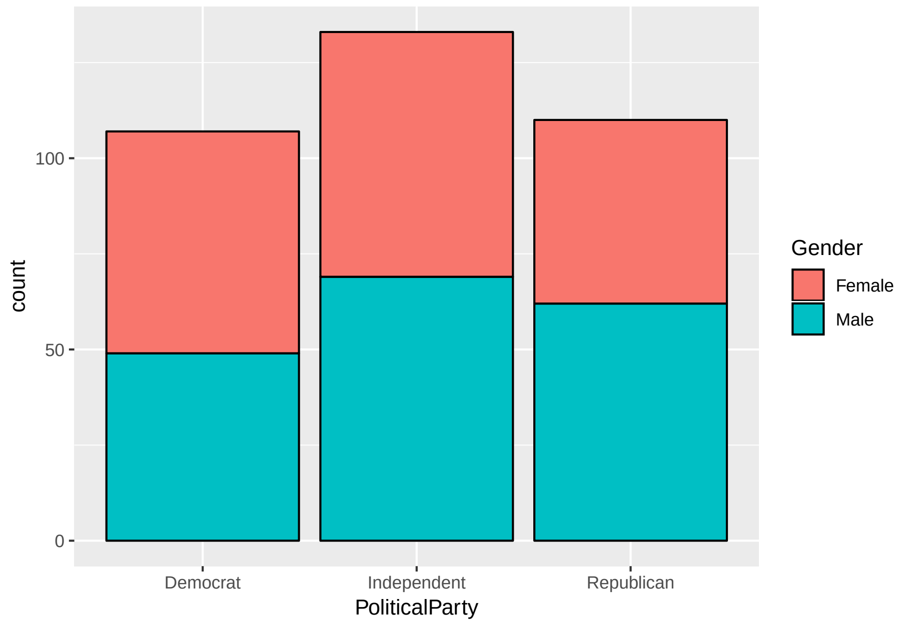
Bar charts are a popular (and horrifically misused way of presenting results). Let's look at a bar chart of number respondents by political affiliation. We have to think a little bit about how this will be presented - consider how a bar chart works. What goes on the x axis and what goes on the y axis?

```r
ggplot(data = Pulse,  aes(x = PoliticalParty))+
  geom_bar()
```



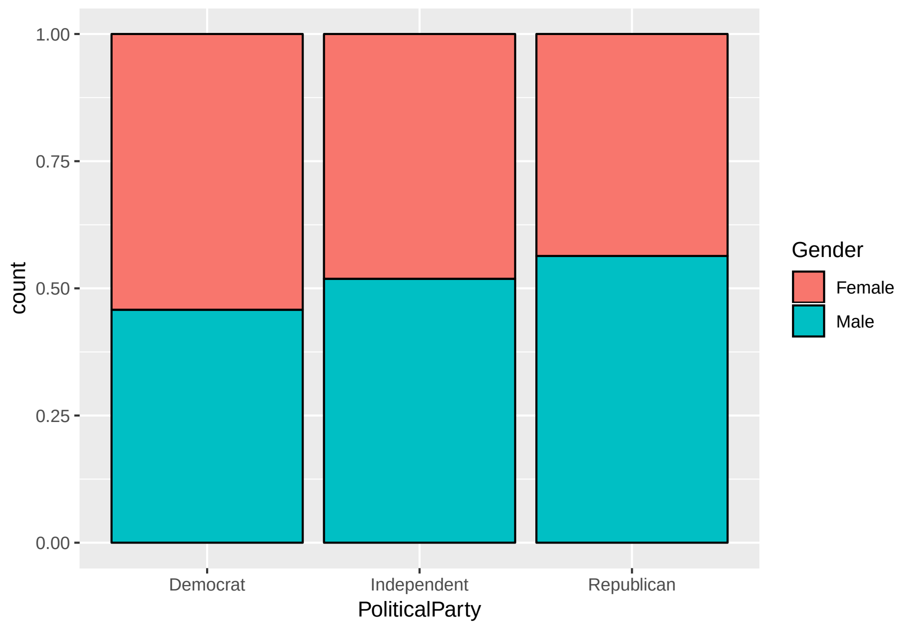
To bring in a second variable then you can map this to the colour or fill aesthetics.

```r
ggplot(data = Pulse,  aes(x = PoliticalParty,fill=Gender))+
  geom_bar(colour="black")
```



It’s hard to make the direct comparison across our three parties - so we can convert to proportions use position=“fill”

```
ggplot(data = Pulse,  aes(x = PoliticalParty,fill=Gender))+
  geom_bar(colour="black",position="fill")
```



QUESTION: Try removing `colour="black"` and describe the difference

QUESTION: Make an (informative) bar chart of level of support for Donald Trump by Gender

```
ggplot(data = Pulse,  aes())+
  geom_bar()
```

4.11 Comparing attractiveness across gender

How can we compare attractiveness across gender in a better way than colouring the points? There are lots of other geoms which might help here. We could produce a boxplot to summarise the level of attractiveness by gender.

QUESTION: Find the geom function for a boxplot and complete the code below to produce a boxplot of attractiveness by gender

```
ggplot(data = Pulse,  aes(x = ?????, y = ?????))+
  geom_?????()
```

The labelling on the y axis is kind of ugly here. Not many people work in base 2.5; but R makes some assumptions on how to label the plots so that it can fit in an 'optimal' number of labels. Let's use scale_y_continuous() to modify this so that all points between 1 and 10 are labelled:

```
ggplot(data = Pulse,  aes(x = ?????, y = ?????))+
  geom_?????()+
    scale_y_continuous(breaks=1:10)
```

Violin plots are also a really nice way of summarising data in a similar way to boxplots ##Do we want to remove the answer to the earlier questions?

```
ggplot(data = Pulse,  aes(x = Gender, y = Attractiveness))+
  geom_violin()+
    scale_y_continuous(breaks=1:10)
```

QUESTION: Modify the following code so that the violins are shaded in different colours for men and women. (Hint: You might need to consult the help menu to look for a new aesthetic we haven't mentioned yet

```
ggplot(data = Pulse,  aes(x = Gender, y = Attractiveness))+
  geom_violin()+
    scale_y_continuous(breaks=1:10)
```

4.12 Splitting plots using Facets

We've seen how to add variables to a plot using aesthetics. Another way, particularly useful for categorical variables, is to split your plot into facets, which are subplots that each display one subset of the data.

```
ggplot(data = Pulse,  aes(x = Gender, y = Attractiveness,fill=Gender))+
  geom_violin()+
    scale_y_continuous(breaks=1:10) +
      facet_wrap(~PoliticalParty)
```

Be careful not to miss the ~ before country. This is used to define a formula which can have something on both sides of ~ in general. It's used in many R function, and you'll see this more later in modelling.

4.13 Themes and labels

Themes refers to the non data elements of your plot. ggplot2 has a huge number of theme options that allows you to control almost every aspect of your plot which becomes important when you want to present or publish your graphs.

The labs() function allows us to label any of our variables and create a title. We only need to change options that we are unhappy with.

```
ggplot(data = Pulse,  aes(x = Gender, y = Attractiveness,fill=Gender))+
  geom_violin()+
    scale_y_continuous(breaks=1:10) +
     facet_wrap(~PoliticalParty)+
      labs(x="Gender",y = "Perceived Physical Attractivess",colour="Gender (again)",ti
```

theme() allows us to modify the size, font, face of labels and the appearance of many other attributes of the plot which do not depend on the data.

```
?theme
```

You will see a a huge number of arguments. You don't need to learn them all, but you can find out about them on that page. Let's say we want to edit the text on the x axis, which is the axis.text.x argument. We can modify different properties of the text through the element_text() function. Let's look at that.

```
?element_text
```

We can see it has arguments such as family (for font), colour, size and angle etc.

QUESTION: Investigate the element_text() function to make the text labels on the x axis large and bold

```
ggplot(data = Pulse,  aes(x = Gender, y = Attractiveness,fill=Gender))+
  geom_violin()+
    scale_y_continuous(breaks=1:10) +
     facet_wrap(~PoliticalParty)+
      labs(x="Gender",y = "Perceived Physical Attractivess",colour="Gender (again)",ti
       theme(axis.text.x = element_text(????))
```

4.14 Recap Exercises

Fix the error(s) in this code to produce a boxplot of attractiveness by age group

```
ggplot(data = Pulse,  aes(x = agegroup, y = attractiveness))+
  geom_boxplot()
```

Fill in the blanks to produce a scatter plot of attractiveness (y axis) by age (x axis)

```
ggplot(data = Pulse,  aes(??????))+
  geom_????()
```

Continuing the previous example, modify the scale on the y axis to show every number between 1 and 10 and the x axis to show every 20 years

```
ggplot(data = Pulse,  aes(??????))+
  geom_????()
```

Produce boxplots of income by gender, with seperate panels for each age group. Use some colours, set a log axis for income, and add sensible titles and labels

```
ggplot(data = Pulse, ...
```

Produce a graph to investigate the is a relationship between income and education; and the extent to which this varies by gender.

Chapter 5

Data Manipulation with dplyr

The previous chapter introduced `ggplot2` which expanded our Base-R vocabulary to help us visualise our data easier.

Now, what if our dataset isn't organised quite the way we want it to create a plot? Maybe it's missing a column we need? Maybe missing values are encoded stragangely? Perhaps we need to standarise the units in a column?

We can use another package to solve these problems `-` `dplyr`! (Wickham et al., 2019b)

Like `ggplot2`, it adds more words to our R vocabulary and is focused on organising your data. `dplyr` is short for "data frame plier". Data frames are an object type in `R` but we don't think it is useful to discuss object types in this tutorial. `data frames` are essentially tables that stores your data in R.

5.1 Set Up for Session

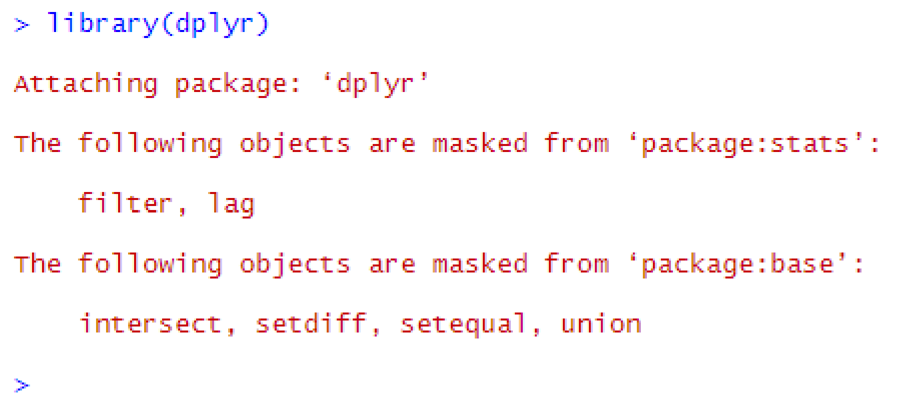
Similarly to the previous chapter, it is assumed you have R Studio running and have installed `ggplot2` and `dplyr`. If not, run the following code;

```r
install.packages(c("ggplot2","dplyr"))
```

We begin by loading `dplyr` and `ggplot2`.

```r
library(dplyr)
library(ggplot2)
```

When you load `dplyr` you will get a warning message;



```
> library(dplyr)

Attaching package: ‘dplyr’

The following objects are masked from ‘package:stats’:

    filter, lag

The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union

>
```

Don’t worry!.

What it is saying is that `dplyr` has a few functions with the same name as built-in functions, so it will now use the ones from `dplyr` by default when these are called.

QUESTION: Can you match the interpretation of the error with what is written in the error message? Why do you think they haved used the phrase ‘masked’?

```r
library(dplyr)
library(ggplot2)
```

Let’s start by loading the miniIMDB dataset. This process is that same as described in the previous chapter but loading `04-miniImdb.RData` instead of `03 Pulse.RData`.

5.2 Description of Datasets Being Used

The “imdb” dataset has been built from the subsets of the Internet Movie Database made available for non-commercial purposes by the IMDb team: IMDB (2018)

It contains the following informations for all the entries having more than 500 votes, that are not of type “tvEpisodes” and for which information about year of release, running time and director(s) was available at the time of extraction (24/10/2018):

Variable	Meaning
title	the popular title of the entry
type	type of the entry: movie short, tvMiniSeries, tvMovie, tvSeries, tvShort, tvSpecial, video or videoGame
year	the year of release or start of release for series
length	the duration of the running time in minutes
director	the director (or director appearing first in the list of directors)
birthYear	year of birth of director
NumVotes	number of votes for the entry
averageRating	IMDb's weighted average rating for the entry

As well as a list of logical (TRUE/FALSE) columns qualifying the genre of the entry:

Genres
animation
action
adventure
comedy
documentary
fantasy
romance
sci_fi
thriller

The "miniImdb" dataset is a small subset of "imdb". It contains the following 5 first variables of "imdb" for the titles that have received more than 1 million votes:

miniImdb
title
type
year
length
numVotes

We will start by using the miniImdb dataset but we will later move to the full imdb dataset when we get more familiar with data manipulation.

5.3 Slicing, Filter and Select

We can use square brackets as a 'quick' way to get subsets of datasets (also called slicing) based on position;

DATASETNAME[ROW NUMBERS,COLUMN NUMBERS]

```
miniImdb[1,]
```

```
## # A tibble: 1 x 5
##   title         type   year length numVotes
##   <chr>         <fct> <dbl>  <dbl>    <int>
## 1 The Godfather movie  1972    175  1374861
```

```
miniImdb[,1]
```

```
## # A tibble: 29 x 1
##    title
##    <chr>
##  1 The Godfather
##  2 Star Wars: Episode IV - A New Hope
##  3 Star Wars: Episode V - The Empire Strikes Back
##  4 The Silence of the Lambs
##  5 Schindler's List
##  6 Forrest Gump
##  7 Pulp Fiction
##  8 The Shawshank Redemption
##  9 Se7en
## 10 The Lord of the Rings: The Fellowship of the Ring
## # ... with 19 more rows
```

```
miniImdb[1:5,1:2]
```

```
## # A tibble: 5 x 2
##   title                                          type
##   <chr>                                          <fct>
## 1 The Godfather                                  movie
## 2 Star Wars: Episode IV - A New Hope             movie
## 3 Star Wars: Episode V - The Empire Strikes Back movie
## 4 The Silence of the Lambs                       movie
## 5 Schindler's List                               movie
```

But to get more useful subsets we would probably need to base these on characteristics or names rather than positions. We can use the `filter()` and `select()` functions for that from `dplyr`.

filter() helps you get the rows you are interested in

```r
filter(miniImdb, type=="tvSeries")
```

```r
filter(miniImdb, year>2005)
```

```
## # A tibble: 2 x 5
##   title           type      year length numVotes
##   <chr>           <fct>    <dbl>  <dbl>    <int>
## 1 Breaking Bad    tvSeries  2008     49  1124817
## 2 Game of Thrones tvSeries  2011     57  1365039
```

```
## # A tibble: 12 x 5
##    title                 type      year length numVotes
##    <chr>                 <fct>    <dbl>  <dbl>    <int>
##  1 Inglourious Basterds  movie     2009    153  1069563
##  2 The Departed          movie     2006    151  1031834
##  3 The Dark Knight       movie     2008    152  1975810
##  4 The Prestige          movie     2006    130  1020039
##  5 Avatar                movie     2009    162  1005456
##  6 Interstellar          movie     2014    169  1218930
##  7 The Avengers          movie     2012    143  1129041
##  8 Breaking Bad          tvSeries  2008     49  1124817
##  9 Game of Thrones       tvSeries  2011     57  1365039
## 10 The Dark Knight Rises movie     2012    164  1335842
## 11 Inception             movie     2010    148  1755897
## 12 Django Unchained      movie     2012    165  1156987
```

Whereas select helps you keep only the columns that you care about, by listing their names

```r
select(miniImdb,title,year, numVotes)
```

```
## # A tibble: 29 x 3
##    title                                               year numVotes
##    <chr>                                              <dbl>    <int>
##  1 The Godfather                                       1972  1374861
##  2 Star Wars: Episode IV - A New Hope                  1977  1078317
##  3 Star Wars: Episode V - The Empire Strikes Back      1980  1008039
##  4 The Silence of the Lambs                            1991  1075970
##  5 Schindler's List                                    1993  1035642
##  6 Forrest Gump                                        1994  1527656
##  7 Pulp Fiction                                        1994  1566510
##  8 The Shawshank Redemption                            1994  2006753
##  9 Se7en                                               1995  1226169
## 10 The Lord of the Rings: The Fellowship of the Ring  2001  1445423
## # ... with 19 more rows
```

or by going in sequence - from title to year for example

```
select(miniImdb,title:year)
```

```
## # A tibble: 29 x 3
##    title                                         type   year
##    <chr>                                         <fct> <dbl>
##  1 The Godfather                                 movie  1972
##  2 Star Wars: Episode IV - A New Hope            movie  1977
##  3 Star Wars: Episode V - The Empire Strikes Back    movie  1980
##  4 The Silence of the Lambs                      movie  1991
##  5 Schindler's List                              movie  1993
##  6 Forrest Gump                                  movie  1994
##  7 Pulp Fiction                                  movie  1994
##  8 The Shawshank Redemption                      movie  1994
##  9 Se7en                                         movie  1995
## 10 The Lord of the Rings: The Fellowship of the Ring movie  2001
## # ... with 19 more rows
```

So in summary,
`filter()` works by subsetting rows from a dataset
`select()` works for subset of columns

QUESTION: Write the code that would give you all the entries that have been released before 2000

```
filter(miniImdb, ????)
```

QUESTION: What would you do to only keep the columns title, length and numVotes from the miniImdb dataset

```
select(miniImdb,????)
```

5.4 Creating new dataframe

All of these subsets so far have only produced temporary results printed into the console window. We usually subset data so that we can then do something with it later. One option is to assign the subset to a new dataframe with the arrow: "<-". We will learn another way later.

```
recentEntries<-filter(miniImdb, year>2005)
recentEntries
```

```
## # A tibble: 12 x 5
##    title                 type      year length numVotes
##    <chr>                 <fct>    <dbl>  <dbl>    <int>
##  1 Inglourious Basterds  movie     2009    153  1069563
##  2 The Departed          movie     2006    151  1031834
##  3 The Dark Knight       movie     2008    152  1975810
##  4 The Prestige          movie     2006    130  1020039
##  5 Avatar                movie     2009    162  1005456
##  6 Interstellar          movie     2014    169  1218930
##  7 The Avengers          movie     2012    143  1129041
##  8 Breaking Bad          tvSeries  2008     49  1124817
##  9 Game of Thrones       tvSeries  2011     57  1365039
## 10 The Dark Knight Rises movie     2012    164  1335842
## 11 Inception             movie     2010    148  1755897
## 12 Django Unchained      movie     2012    165  1156987
```

```
tvSeriesData<-filter(miniImdb, type=="tvSeries")
tvSeriesData
```

```
## # A tibble: 2 x 5
##   title           type      year length numVotes
##   <chr>           <fct>    <dbl>  <dbl>    <int>
## 1 Breaking Bad    tvSeries  2008     49  1124817
## 2 Game of Thrones tvSeries  2011     57  1365039
```

Note the == sign to mean IS EQUAL TO
A single equals sign is an assignment statement: x=y "set x to be equal y"
A double equals sign is a question x==y "is x equal to y?"
within the filter() function the single = sign will return an error. Not all functions may be so friendly!

```
filter(miniImdb, type="tvSeries")
```

```
## `type` (`type = "tvSeries"`) must not be named, do you need `==`?
```

Remember R is case sensitive, and cannot do any association of meaning on its own

```
filter(miniImdb, type=="tv Series")
```

```
## # A tibble: 0 x 5
## # ... with 5 variables: title <chr>, type <fct>, year <dbl>, length <dbl>,
## #   numVotes <int>
```

nothing

```
filter(miniImdb, type=="tvseries")
```

```
## # A tibble: 0 x 5
## # ... with 5 variables: title <chr>, type <fct>, year <dbl>, length <dbl>,
## #   numVotes <int>
```

also nothing

```
filter(miniImdb, Type=="tvSeries")
```

```
## Error: object 'Type' not found
```

still nothing

We can also produce a subset of a subset:

```
titleVotesRecent<-select(recentEntries, title, numVotes)
titleVotesRecent
```

```
## # A tibble: 12 x 2
##    title                 numVotes
##    <chr>                    <int>
##  1 Inglourious Basterds   1069563
##  2 The Departed           1031834
##  3 The Dark Knight        1975810
##  4 The Prestige           1020039
##  5 Avatar                 1005456
##  6 Interstellar           1218930
##  7 The Avengers           1129041
##  8 Breaking Bad           1124817
##  9 Game of Thrones        1365039
## 10 The Dark Knight Rises  1335842
## 11 Inception              1755897
## 12 Django Unchained       1156987
```

QUESTION: Produce a subset of the data for the type “movie” and assign it to an object called movies

```
??? <- filter(miniImdb, ????==????)
???
```

QUESTION: Is there any movie who received more than 2 million votes?

```
filter(????, ????)
```

5.5 Column Transformations

We can create new columns with the function `mutate()`
For example let's try to convert the length of the entries in hour rather than in minutes

```
mutate(miniImdb, lengthInHour = length/60)
```

```
## # A tibble: 29 x 6
##    title                           type   year length numVotes lengthInHour
##    <chr>                           <fct> <dbl>  <dbl>    <int>        <dbl>
##  1 The Godfather                   movie  1972    175  1374861         2.92
##  2 Star Wars: Episode IV - A New ~ movie  1977    121  1078317         2.02
##  3 Star Wars: Episode V - The Emp~ movie  1980    124  1008039         2.07
##  4 The Silence of the Lambs        movie  1991    118  1075970         1.97
##  5 Schindler's List                movie  1993    195  1035642         3.25
##  6 Forrest Gump                    movie  1994    142  1527656         2.37
##  7 Pulp Fiction                    movie  1994    154  1566510         2.57
##  8 The Shawshank Redemption        movie  1994    142  2006753         2.37
##  9 Se7en                           movie  1995    127  1226169         2.12
## 10 The Lord of the Rings: The Fel~ movie  2001    178  1445423         2.97
## # ... with 19 more rows
```

The usual signs that R uses to make calculations are:

Symbol	Meaning
`*`	multiply
`/`	divide
`+`	add
`-`	substract
`**`	raise to the power

QUESTION: Create a new dataframe adding a column giving the number of votes in million to miniImdb and show only the title and this newly created column

```
miniImdbMillion <- mutate(miniImdb, ???? = ????)
select(????, ????)
```

5.6 More ways to filter

When we look at the full imdb dataset, it will be useful to know that we can use multiple conditions and additional fonctions to filter rows:

Here are the possible logical symbols to use when doing conditions in R:

Code	Meaning
==	EQUALS
&	AND
!	NOT
<	less than
>	more than

And we can combine these together. On the "recentEntries" data let's get all the entries with a length between 1h30 (90min) and 2h30 (150min)

```
filter(recentEntries,length>90 & length<150)
```

```
## # A tibble: 3 x 5
##   title         type   year length numVotes
##   <chr>         <fct> <dbl>  <dbl>    <int>
## 1 The Prestige  movie  2006    130  1020039
## 2 The Avengers  movie  2012    143  1129041
## 3 Inception     movie  2010    148  1755897
```

We can also use functions like `max()` or `min()` to help us with the filtering. For example, maybe we want to know which entrie's length is the largest

```
filter(recentEntries,length==max(length))
```

```
## # A tibble: 1 x 5
##   title         type   year length numVotes
##   <chr>         <fct> <dbl>  <dbl>    <int>
## 1 Interstellar movie  2014    169  1218930
```

Finally, going back to our very first examples, note that the function `slice()` is an alternative way to get specific rows by positions in a dataset. We will use it on the full imdb dataset

```
slice(recentEntries, 1)
```

```
## # A tibble: 1 x 5
##   title                type   year length numVotes
##   <chr>                <fct> <dbl>  <dbl>    <int>
## 1 Inglourious Basterds movie  2009    153  1069563
```

```
slice(miniImdb, 1:5)
```

```
## # A tibble: 5 x 5
##   title                                        type   year length numVotes
##   <chr>                                        <fct> <dbl>  <dbl>    <int>
## 1 The Godfather                                movie  1972    175  1374861
## 2 Star Wars: Episode IV - A New Hope           movie  1977    121  1078317
## 3 Star Wars: Episode V - The Empire Strikes Ba~ movie  1980    124  1008039
## 4 The Silence of the Lambs                     movie  1991    118  1075970
## 5 Schindler's List                             movie  1993    195  1035642
```

QUESTION: Which movie has the oldest year of release among the movies of the miniImdb dataset?

```
filter(miniImdb,????)
```

QUESTION: Which entry has the oldest year of release in the full imdb dataset?

```
filter(????,????)
```

QUESTION: I'm trying to find out which movie has the oldest year of release in the full imdb dataset. Can you guess why this code doesn't work? How should you modify it?

```
filter(imdb, type=="movie" & year==min(year))
```

```
## # A tibble: 0 x 17
## # ... with 17 variables: title <chr>, type <fct>, year <dbl>,
## #   length <dbl>, numVotes <int>, averageRating <dbl>, director <chr>,
## #   birthYear <dbl>, animation <lgl>, action <lgl>, adventure <lgl>,
## #   comedy <lgl>, documentary <lgl>, fantasy <lgl>, romance <lgl>,
## #   sci_fi <lgl>, thriller <lgl>
```

5.7 Aggregation/Grouping

The imdb dataset contains entries of various types, but we only saw entries of type movie and tvSeries so far, because the other types don't have enough votes to be in the miniImdb dataset. So we will now use the full imdb dataset.

One thing that would be interesting is to get the entries with highest number of votes for each type of entry. You know how to do it for each type separately, by filtering on the specific type first:

```
short<-filter(imdb,type=="short")
filter(short, numVotes==max(numVotes))
```

```
## # A tibble: 1 x 17
##   title type   year length numVotes averageRating director birthYear
##   <chr> <fct> <dbl>  <dbl>    <int>         <dbl> <chr>        <dbl>
## 1 Kung~ short  2015     31    50331             8 David S~        NA
## # ... with 9 more variables: animation <lgl>, action <lgl>,
## #   adventure <lgl>, comedy <lgl>, documentary <lgl>, fantasy <lgl>,
## #   romance <lgl>, sci_fi <lgl>, thriller <lgl>
```

But wouldn't it be nice to do it for all types in a single move? We can do so using the function group_by() instead of filter() for the first step

```
imdb_type<-group_by(imdb,type)
filter(imdb_type, numVotes==max(numVotes))
```

```
## # A tibble: 9 x 17
## # Groups:   type [9]
##   title type   year length numVotes averageRating director birthYear
##   <chr> <fct> <dbl>  <dbl>    <int>         <dbl> <chr>        <dbl>
## 1 Eddi~ tvSp~  1983     69    13995           8.2 Bruce G~        NA
## 2 The ~ movie  1994    142  2006753           9.3 Frank D~      1959
## 3 Band~ tvMi~  2001    594   297551           9.5 David F~      1959
## 4 The ~ video  2003    100    66405           7.4 Shin'ic~        NA
## 5 High~ tvMo~  2006     98    70030           5.3 Kenny O~      1950
## 6 Shre~ tvSh~  2007     21    11025           6.5 Gary Tr~      1960
## 7 Game~ tvSe~  2011     57  1365039           9.5 Matt Sh~      1975
## 8 Halo~ vide~  2009     34     3077           7.6 Rich Wi~        NA
## 9 Kung~ short  2015     31    50331           8   David S~        NA
## # ... with 9 more variables: animation <lgl>, action <lgl>,
## #   adventure <lgl>, comedy <lgl>, documentary <lgl>, fantasy <lgl>,
## #   romance <lgl>, sci_fi <lgl>, thriller <lgl>
```

QUESTION: Can you find out which title has the oldest release year for each type of entry?

```
imdb_type<-group_by(imdb,????)
filter(????, ????==????)
```

5.8 Pipes

I'm a big sci-fi fan, and I see that none of the most voted entries are of this genre (see above). Let's use filter to only keep the sci-fi entries:

```
imdbSciFi<-filter(imdb,sci_fi=="TRUE")
imdbSciFi_type<-group_by(imdbSciFi,type)
filter(imdbSciFi_type, numVotes==max(numVotes))
```

```
## # A tibble: 7 x 17
## # Groups:   type [7]
##   title type   year length numVotes averageRating director birthYear
##   <chr> <fct> <dbl>  <dbl>    <int>         <dbl> <chr>        <dbl>
## 1 A Tr~ short  1902     13    36791           8.2 Georges~      1861
## 2 Batt~ tvMo~  2007    101    18920           7.7 FÃ©lix ~      1951
## 3 Robo~ tvSh~  2007     30     7134           8.2 Seth Gr~      1974
## 4 Ince~ movie  2010    148  1755897           8.8 Christo~      1970
## 5 Deat~ video  2010    100    27769           5.6 Roel Re~      1969
## 6 The ~ tvSe~  2010     44   755188           8.4 Lesli L~        NA
## 7 11.2~ tvMi~  2016     60    56331           8.2 James S~        NA
## # ... with 9 more variables: animation <lgl>, action <lgl>,
## #   adventure <lgl>, comedy <lgl>, documentary <lgl>, fantasy <lgl>,
## #   romance <lgl>, sci_fi <lgl>, thriller <lgl>
```

To get from the original data to this output we had to create a number of intermediate steps
imdb -> filter on sci-fi -> group_by type -> filter to max
We either need to explicitly save each of the steps as a dataeframe OR we can be clever and use pipes '%>%

```
imdb %>%
  filter(sci_fi=="TRUE") %>%
    group_by(type) %>%
      filter(numVotes==max(numVotes))
```

```
## # A tibble: 7 x 17
## # Groups:   type [7]
##   title type   year length numVotes averageRating director birthYear
##   <chr> <fct> <dbl>  <dbl>    <int>         <dbl> <chr>        <dbl>
## 1 A Tr~ short  1902     13    36791           8.2 Georges~      1861
## 2 Batt~ tvMo~  2007    101    18920           7.7 FÃ©lix ~      1951
## 3 Robo~ tvSh~  2007     30     7134           8.2 Seth Gr~      1974
## 4 Ince~ movie  2010    148  1755897           8.8 Christo~      1970
## 5 Deat~ video  2010    100    27769           5.6 Roel Re~      1969
## 6 The ~ tvSe~  2010     44   755188           8.4 Lesli L~        NA
## 7 11.2~ tvMi~  2016     60    56331           8.2 James S~        NA
## # ... with 9 more variables: animation <lgl>, action <lgl>,
## #   adventure <lgl>, comedy <lgl>, documentary <lgl>, fantasy <lgl>,
## #   romance <lgl>, sci_fi <lgl>, thriller <lgl>
```

With the piping system we start with the name of the data (imdb). This then gets carried through in each step - the data from the end of line 1 gets automatically used in line 2 and so on.

We do not need to call the dataframe name as the first option in the functions that we use

And we can easily add on extra functions. Select() might be useful to have a quicker look at all the information we are interested in.

```
imdb %>%
  filter(sci_fi=="TRUE") %>%
    group_by(type) %>%
      filter(numVotes==max(numVotes)) %>%
        select(title,type,averageRating,numVotes)
```

```
## # A tibble: 7 x 4
## # Groups:   type [7]
##   title                        type         averageRating numVotes
##   <chr>                        <fct>                <dbl>    <int>
## 1 A Trip to the Moon           short                  8.2    36791
## 2 Battlestar Galactica: Razor tvMovie                7.7    18920
## 3 Robot Chicken: Star Wars     tvShort                8.2     7134
## 4 Inception                    movie                  8.8  1755897
## 5 Death Race 2                 video                  5.6    27769
## 6 The Walking Dead             tvSeries               8.4   755188
## 7 11.22.63                     tvMiniSeries           8.2    56331
```

QUESTION: Can you find out for each type of entry, which is the best rated title using pipes?

```
imdb %>%
  group_by(????) %>%
    filter(????==????)
```

5.9 The summarise() function

To know the average rating of all the movies, we could use the function summarize:

```
imdb %>%
  filter(type=="movie") %>%
    summarize(mean=mean(averageRating))
```

```
## # A tibble: 1 x 1
##    mean
##   <dbl>
## 1  6.32
```

But we can get lot's of other summary information, like the number of entries, using the function n() the standard deviation of the ratings using the function sd() or the average number of votes

```
imdb %>%
  filter(type=="movie") %>%
    summarize(n=n(), meanRating=mean(averageRating), sdRating=sd(averageRating), meanVotes=mean(n
```

```
## # A tibble: 1 x 4
##       n meanRating sdRating meanVotes
##   <int>      <dbl>    <dbl>     <dbl>
## 1 39221       6.32     1.20    17775.
```

And we can combine it with group_by to have these information for both the action and the non action movies

```
imdb %>%
  filter(type=="movie") %>%
    group_by(action) %>%
      summarize(n=n(), meanRating=mean(averageRating), sdRating=sd(averageRating), meanVotes=mean
```

```
## # A tibble: 2 x 5
##   action     n meanRating sdRating meanVotes
##   <lgl>  <int>      <dbl>    <dbl>     <dbl>
## 1 FALSE  32811       6.39     1.17    14898.
## 2 TRUE    6410       5.97     1.31    32499.
```

by the way, you can group by more than one variable

```
imdb %>%
  filter(type=="movie") %>%
    group_by(action, adventure, comedy, sci_fi) %>%
      summarize(n = n(), meanRating=mean(averageRating), sdRating=sd(averageRating), m
```

```
## # A tibble: 15 x 8
## # Groups:   action, adventure, comedy [8]
##    action adventure comedy sci_fi     n meanRating sdRating meanVotes
##    <lgl>  <lgl>     <lgl>  <lgl>  <int>      <dbl>    <dbl>     <dbl>
##  1 FALSE  FALSE     FALSE  FALSE  18324       6.53     1.15    13353.
##  2 FALSE  FALSE     FALSE  TRUE    1020       5.35     1.41    24846.
##  3 FALSE  FALSE     TRUE   FALSE  10964       6.27     1.11    12320.
##  4 FALSE  FALSE     TRUE   TRUE     256       5.64     1.25    14883.
##  5 FALSE  TRUE      FALSE  FALSE   1264       6.49     1.10    27982.
##  6 FALSE  TRUE      FALSE  TRUE     107       5.55     1.60    62676.
##  7 FALSE  TRUE      TRUE   FALSE    850       6.27     1.16    41371.
##  8 FALSE  TRUE      TRUE   TRUE      26       6.13     1.39   102517.
##  9 TRUE   FALSE     FALSE  FALSE   3210       6.09     1.24    19461.
## 10 TRUE   FALSE     FALSE  TRUE     430       5.11     1.43    44512.
## 11 TRUE   FALSE     TRUE   FALSE   1058       6.03     1.15    17225.
## 12 TRUE   FALSE     TRUE   TRUE      34       4.97     1.34    27008.
## 13 TRUE   TRUE      FALSE  FALSE   1184       6.03     1.37    54860.
## 14 TRUE   TRUE      FALSE  TRUE     172       5.86     1.48   180236.
## 15 TRUE   TRUE      TRUE   FALSE    322       5.71     1.44    36063.
```

“‘ ARRANGE, UNGROUP, SLICE INTRODUCED LATER Let’s see what’s the lowest rated combination of genres.

QUESTION: Can you find the lowest rated combination of genres from above? HINT: This can be done with arrange(), ungroup() and slice()

```
imdb %>%
  filter(type=="movie") %>%
    group_by(action, adventure, comedy, sci_fi) %>%
      summarize(n = n(), meanRating=mean(averageRating), sdRating=sd(averageRating), m
        arrange(meanRating) %>%
          ungroup() %>%
          slice(1)
```

```
## # A tibble: 1 x 8
##   action adventure comedy sci_fi     n meanRating sdRating meanVotes
##   <lgl>  <lgl>     <lgl>  <lgl>  <int>      <dbl>    <dbl>     <dbl>
## 1 TRUE   FALSE     TRUE   TRUE      34       4.97     1.34    27008.
```

Oh boy, you probably don't want to watch an action/comedy/sc-fi movie that wouldn't be of genre adventure!

QUESTION: Can you calculate for each year, the average length and rating of the released movies? Save the result in a dataframe

```
???? <- imdb %>%
  filter(????) %>%
    group_by(????) %>%
      summarize(????)
```

"'

QUESTION: Would you manage to plot the average length of movies (as y) against the year of release (as x)?

```
ggplot(data = ????, aes(y=????, x=????)) +
  geom_????
```

5.10 Arranging

Ok what we probably want at this point is think about the movie we are going to watch tonight.

The best rated sci-fi movie is:

```
imdb %>%
  filter(type=="movie" & sci_fi=="TRUE") %>%
    filter(averageRating==max(averageRating))
```

```
## # A tibble: 1 x 17
##   title type   year length numVotes averageRating director birthYear
##   <chr> <fct> <dbl>  <dbl>    <int>         <dbl> <chr>        <dbl>
## 1 Ince~ movie  2010    148  1755897           8.8 Christo~      1970
## # ... with 9 more variables: animation <lgl>, action <lgl>,
## #   adventure <lgl>, comedy <lgl>, documentary <lgl>, fantasy <lgl>,
## #   romance <lgl>, sci_fi <lgl>, thriller <lgl>
```

But I already watched it

Maybe, I should check the first 10 sci-fi movies in the dataset using `slice()`:

```
imdb %>%
  filter(type=="movie" & sci_fi=="TRUE") %>%
    slice(1:10)
```

```
## # A tibble: 10 x 17
##    title type   year length numVotes averageRating director birthYear
##    <chr> <fct> <dbl>  <dbl>    <int>         <dbl> <chr>        <dbl>
##  1 20,0~ movie  1916    105     1342           7   Stuart ~      1883
##  2 Dr. ~ movie  1920     49     4148           7   John S.~      1878
##  3 The ~ movie  1925    106     4015           7.1 Harry O~      1885
##  4 Metr~ movie  1927    153   136815           8.3 Fritz L~      1890
##  5 Fran~ movie  1931     70    56442           7.9 James W~      1889
##  6 Dr. ~ movie  1931     98    10969           7.7 Rouben ~      1897
##  7 The ~ movie  1932     68     2694           6.4 Charles~      1900
##  8 The ~ movie  1933     71    24943           7.7 James W~      1889
##  9 Isla~ movie  1932     70     6863           7.5 Erle C.~      1896
## 10 King~ movie  1933    100    71884           7.9 Ernest ~      1893
## # ... with 9 more variables: animation <lgl>, action <lgl>,
## #   adventure <lgl>, comedy <lgl>, documentary <lgl>, fantasy <lgl>,
## #   romance <lgl>, sci_fi <lgl>, thriller <lgl>
```

We only get very old movies, because the dataset seems to be somewhat arranged by year of release. So before using slice(), we want to arrange the movies by rating using the function `arrange()`:

```
imdb %>%
  filter(type=="movie" & sci_fi=="TRUE") %>%
    arrange(averageRating) %>%
      slice(1:10)
```

```
## # A tibble: 10 x 17
##    title type   year length numVotes averageRating director birthYear
##    <chr> <fct> <dbl>  <dbl>    <int>         <dbl> <chr>        <dbl>
##  1 Brow~ movie  2015     98      712           1.2 Francis~        NA
##  2 Purge movie  2010     80     1131           1.4 David K~        NA
##  3 Nukie movie  1987     95     1021           1.5 Michael~      1950
##  4 Star~ movie  2009     81      511           1.5 Jon Bon~        NA
##  5 Ultr~ movie  1990     81      676           1.6 Kevin T~        NA
##  6 Alie~ movie  2011     80     1523           1.6 Lewis S~        NA
##  7 Evil~ movie  2006     90      578           1.7 Jim Car~        NA
##  8 Univ~ movie  2007     85     1383           1.7 Griff F~      1981
##  9 Atla~ movie  2018     86      503            1.7 Jared C~        NA
## 10 Turk~ movie  2006    110    14830            1.9 Kartal ~      1938
## # ... with 9 more variables: animation <lgl>, action <lgl>,
## #   adventure <lgl>, comedy <lgl>, documentary <lgl>, fantasy <lgl>,
## #   romance <lgl>, sci_fi <lgl>, thriller <lgl>
```

Look at the ratings! These are probably the worst movies ever. `arrange()` by default sorts in ascending order, so we need to arrange by descending order of rating:

```
imdb %>%
  filter(type=="movie" & sci_fi=="TRUE") %>%
    arrange(desc(averageRating)) %>%
      slice(1:10)
```

```
## # A tibble: 10 x 17
##    title type   year length numVotes averageRating director birthYear
##    <chr> <fct> <dbl>  <dbl>    <int>         <dbl> <chr>        <dbl>
##  1 Ince~ movie  2010    148  1755897           8.8 Christo~      1970
##  2 The ~ movie  1999    136  1439664           8.7 Lana Wa~      1965
##  3 Inte~ movie  2014    169  1218930           8.6 Christo~      1970
##  4 Alien movie  1979    116   684793           8.5 Ridley ~      1937
##  5 Back~ movie  1985    116   888217           8.5 Robert ~      1951
##  6 Term~ movie  1991    137   870521           8.5 James C~      1954
##  7 The ~ movie  2006    130  1020039           8.5 Christo~      1970
##  8 Alie~ movie  1986    137   579233           8.4 James C~      1954
##  9 Lucia movie  2013    135    10022           8.4 Pawan K~        NA
## 10 Metr~ movie  1927    153   136815           8.3 Fritz L~      1890
## # ... with 9 more variables: animation <lgl>, action <lgl>,
## #   adventure <lgl>, comedy <lgl>, documentary <lgl>, fantasy <lgl>,
## #   romance <lgl>, sci_fi <lgl>, thriller <lgl>
```

That's much better.

QUESTION: What are the 5 best rated thriller tvSeries having more than 10000 votes

```
imdb %>%
  filter(????) %>%
    arrange(????) %>%
      slice(????)
```

5.11 Summary

Key Functions learned:
`filter()`
`select()`
`mutate()`
`group_by()`
`summarise()`
`arrange()`

SUGGESTION: Write down in your own words what each of these functions does

Other concepts covered:
logical operators: `!` `|` `&` `==` numerical operators: `+` `-` `/` `*` pipes`%>%` other useful functions: `slice()` `max()` `min()` `n()` `mean()` `sd()`

Lets combine all of these concepts together into a single piece of R code!:

```
imdb %>%
  group_by(director) %>%
    summarize(n=n(), meanRating=mean(averageRating), sumVotes=sum(numVotes), medianLeng
      mutate(meanVotes=sumVotes/n) %>%
        filter(n>10 & meanVotes>10000) %>%
          arrange(desc(meanRating)) %>%
            select(director, n, meanRating, medianLength)%>%
              slice(1:10)
```

```
## # A tibble: 10 x 4
##    director                 n meanRating medianLength
##    <chr>                <int>      <dbl>        <dbl>
##  1 Christopher Nolan       12       8.09         124
##  2 Hayao Miyazaki          14       8            100.
##  3 Spike Jonze             11       7.8           28
##  4 Quentin Tarantino       12       7.79         145
##  5 David Fincher           12       7.72         128
##  6 Anurag Kashyap          14       7.67         136.
##  7 Akira Kurosawa          30       7.67         114.
##  8 S.S. Rajamouli          11       7.62         158
##  9 William Wyler           28       7.62         106
## 10 Krzysztof Kieslowski    16       7.61          95
```

QUESTION: Write down what this code does

QUESTION: Which line could we remove by making a little modification inside the function summarize()?

QUESTION: Did you notice the use of the function median()? Why would one prefer to use the median rather than the mean?

BONUS CHALLENGE: Could you find among the movie entries, the 10 directors having directed their 5th movie at the youngest age? Who is the 9th one in the list? (hint: you may need to use a function that we haven't seen yet: ungroup())

```
imdb %>%
  filter(????) %>%
    mutate(ageAtRelease=????) %>%
      group_by(????) %>%
        arrange(????) %>%
          slice(????) %>%
            ungroup() %>%
              arrange(????) %>%
                select(????) %>%
                  slice(????)
```

Chapter 6

Importing Data

6.1 Placeholder

Importing data in R can be really easy. To import data in R, you usually use a function with the general form `read.___()` and that's it! To make the dataset accessible in R you have to assign it a name e.g.

```
NAME <- read.___()
```

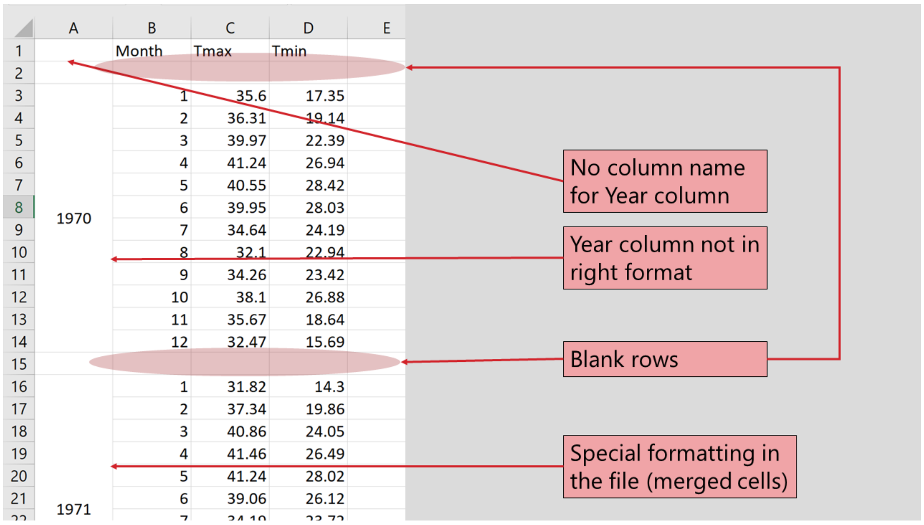
The object 'NAME' is a type of `data.frame`. As mentioned in earlier, it is essentially a large table, so R expects a "complete" table as the input. If the input is not a "complete" table, R will do it's best to fill in the blanks - maybe not in the way you expect!

By "complete" table, we mean;

- Each column is of the same length
- Each column is has the same type of value in each object.
- Usually the first row is are the column names
- Contain no blank rows in the middle of the dataset
- have no special formatting

Depending on your data, one of the best tools to check and edit your data is Excel. Another thing to note is an Excel file should only have one rectangle of data per sheet.

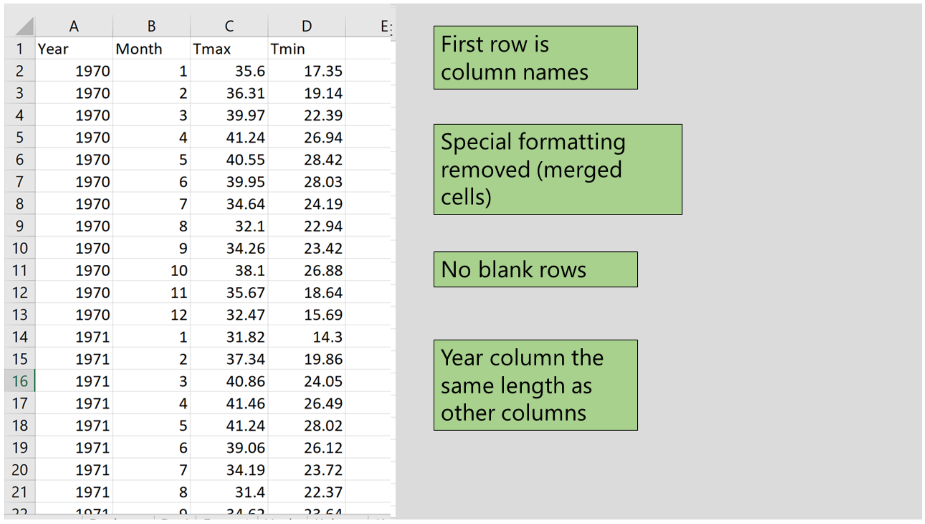
Here is an example of a poorly formatted excel sheet



	A	B	C	D	E
1		Month	Tmax	Tmin	
2					
3		1	35.6	17.35	
4		2	36.31	19.14	
5		3	39.97	22.39	
6		4	41.24	26.94	
7		5	40.55	28.42	
8	1970	6	39.95	28.03	
9		7	34.64	24.19	
10		8	32.1	22.94	
11		9	34.26	23.42	
12		10	38.1	26.88	
13		11	35.67	18.64	
14		12	32.47	15.69	
15					
16		1	31.82	14.3	
17		2	37.34	19.86	
18		3	40.86	24.05	
19		4	41.46	26.49	
20		5	41.24	28.02	
21	1971	6	39.06	26.12	

No column name for Year column
Year column not in right format
Blank rows
Special formatting in the file (merged cells)

and here is the correted version



	A	B	C	D	E
1	Year	Month	Tmax	Tmin	
2	1970	1	35.6	17.35	
3	1970	2	36.31	19.14	
4	1970	3	39.97	22.39	
5	1970	4	41.24	26.94	
6	1970	5	40.55	28.42	
7	1970	6	39.95	28.03	
8	1970	7	34.64	24.19	
9	1970	8	32.1	22.94	
10	1970	9	34.26	23.42	
11	1970	10	38.1	26.88	
12	1970	11	35.67	18.64	
13	1970	12	32.47	15.69	
14	1971	1	31.82	14.3	
15	1971	2	37.34	19.86	
16	1971	3	40.86	24.05	
17	1971	4	41.46	26.49	
18	1971	5	41.24	28.02	
19	1971	6	39.06	26.12	
20	1971	7	34.19	23.72	
21	1971	8	31.4	22.37	

First row is column names
Special formatting removed (merged cells)
No blank rows
Year column the same length as other columns

6.2 Introduction to UKIreland dataset

The UKIreland data contains data from 2000 to 2016 with the gdp, population, and unemployment rate of the UK and Ireland. (FAOSTAT, 2018)

6.3 Reading in data from CSV

CSV format is the most common formats to save data. See notes in the slides about good practice for data structures before reading anything in.

If there are problems in your data before you read it into R, there will be problems with your analysis in R. Make sure you check your data when it comes in - solving the problems in your raw data files is usually easier than solving the problems using R, and will also prevent others using the data from running into the same problems in the future.

```
UKIreland<-read.csv("path/to/file")
```

6.4 Reading in from Excel

Reading from excel is also easy; but needs an extra package to be installed. There are a lot of different options to do this! My preferred choice is to use the openxlsx library.

```
library(openxlsx)

UKIreland2<-read.xlsx("path/to/UKIreland.xlsx")

UKIreland2
```

Note that the excel version of the file is different; there is one sheet for each country.

The read.xlsx function automatically only reads in the first sheet. So this line above has only read in the data for Ireland, from the first tab in the Excel file. However, if we have named sheets in our Excel file then reading in the sheets we want is easy:

```
UK<-read.xlsx("path/to/UKIreland.xlsx","UK")
Ireland<-read.xlsx("path/to/UKIreland.xlsx","Ireland")
```

We can use `filter()` to produce subsets of our data - e.g. to find the maximum unemployment rate:

```
filter(UKIreland,unemployment==max(unemployment))
```

```
##   country date      gdp population unemployment capital
## 1 Ireland 2012 49177.44    4586897       14.725  Dublin
```

Remember the double == when making logical statements

Using pipes and group_by() lets us combine multiple tasks together easily - e.g. finding the maximum unemployment rate within each country

```
UKIreland %>%
  group_by(country) %>%
    filter(unemployment==max(unemployment))
```

```
## # A tibble: 2 x 6
## # Groups:   country [2]
##   country         date    gdp population unemployment capital
##   <fct>          <int>  <dbl>      <int>        <dbl> <fct>
## 1 Ireland         2012 49177.    4586897        14.7  Dublin
## 2 United Kingdom  2011 41412.   63258918         8.09 London
```

Or we could use summarise() to calculate average unemployment rates per country:

```
UKIreland %>%
  group_by(country) %>%
    summarise(average_unemployment=mean(unemployment))
```

```
## # A tibble: 2 x 2
##   country        average_unemployment
##   <fct>                         <dbl>
## 1 Ireland                        8.18
## 2 United Kingdom                 6.02
```

Note this time - only a single = sign in summarise because we are assigning a value (set average_unemployment to be equal to mean of unemployment). Earlier we wanted R to check if two things were equal (when is unemployment equal to maximum unemployment), so we used `==`.

mutate() is a nice function to help us create new variables. E.g. the total number of unemployed people

```
mutate(UKIreland,TotalUnemployed=unemployment*population/100)
```

country	date	gdp	population	unemployment	capital	TotalUnemployed
Ireland	2000	26241.51	3805174	4.266667	Dublin	162354.1
Ireland	2001	28227.28	3866243	3.925000	Dublin	151750.0
Ireland	2002	32539.95	3931947	4.491667	Dublin	176610.0
Ireland	2003	41107.03	3996521	4.616667	Dublin	184506.1
Ireland	2004	47630.93	4070262	4.491667	Dublin	182822.6
Ireland	2005	50878.64	4159914	4.400000	Dublin	183036.2
Ireland	2006	54306.91	4273591	4.525000	Dublin	193380.0
Ireland	2007	61359.64	4398942	4.691667	Dublin	206383.7
Ireland	2008	61257.90	4489544	6.433333	Dublin	288827.3
Ireland	2009	52104.04	4535375	12.050000	Dublin	546512.7

Note that this is obviously a stupid number from a methodological point of view since unemployment rates are based on “eligible workforce population” rather than total population. But I’m not an economist so let’s go with that for now.

Chapter 7

Merging datasets

7.1 Set Up for Session

In this session, we are going to use the following packages; `dplyr, ggplot2, openxlsx, tidyr, plotly`. If you get an error saying `there is no package called _____`, install the package as covered in the first tutorial.

```
library(dplyr)
library(ggplot2)
library(openxlsx)
library(tidyr)
library(plotly)
```

We have used the following datasets in this tutorial. Import them as covered in the previous tutorial `UKIreland.csv, UKIreland.xlsx, UN_LivestockData.csv, WorldBankData.csv`

7.2 Binding

After sucessfully loading the datasets, there should be seperate data frames of the UK data and Ireland data that we used in the previous tutorial. Ideally we would like to combine these together into a single data frame.

In this case the two data frames have the same structure; the columns are the same, the column names are the same and there are the same number of rows (one for each year).

We can use the `rbind()` function to merge the data frames together. `rbind()` attatches the rows of the second data frame below the first data frame.

```
UKIreland<-rbind(UK,Ireland)
```

`rbind()` only works because the UK data has the same columns as the Ireland data. For example, we will get an error if we had calculated a new column in one dataset but not in the other:

```
UK2 <- mutate(UK,pop_mill=population/1000000)

rbind(UK2,Ireland)
```

```
## Error in rbind(deparse.level, ...): numbers of columns of arguments do not match
```

There is also a function called `cbind`. This binds the columns together.

```
UKIreland2<-cbind(UK,Ireland)
```

`cbind()` will only work if we have the same number of rows in each data. cbind also produces a slightly confusing dataset - we now have two sets columns called "population", "country" and so on; this makes it very difficult to work with!

`rbind()` is used often but `cbind()` is not since to be useful the two datasets need to have the same number of rows with each row describing the same thing. There are more functions to allow more flexible types of merging of datasets.

7.3 Introducing merging

`dplyr` has a number of clever functions for merging data.

Join function	**Description**
`full_join()`	merges two datasets and keeping all observations from both datasets
`inner_join()`	merges two datasets and only keeping the matching observations
`left_join()`	merges two datasets and keeping all observations from the first ("left") data set but only the observations that match from the second ("right") data set
`right_join()`	merges two datasets and keeping all observations from the second ("right") data set but only the observations that match from the first ("left") data set
`anti_join()`	identifying rows present in the first ("left") dataset which do not have a match in the second ("right") dataset.

We will work through these sequentially. The function for merging all the data is full_join. The key identifier column between UK and Ireland would be the “date” column.

```r
full_join(UK,Ireland,by="date")
```

```
##         country.x date    gdp.x population.x unemployment.x capital.x
## 1  United Kingdom 2000 27982.36     58892514       5.450000    London
## 2  United Kingdom 2001 27427.59     59119673       5.083333    London
## 3  United Kingdom 2002 29785.99     59370479       5.175000    London
## 4  United Kingdom 2003 34173.98     59647577       5.008333    London
## 5  United Kingdom 2004 39983.98     59987905       4.750000    London
## 6  United Kingdom 2005 41732.64     60401206       4.841667    London
## 7  United Kingdom 2006 44252.32     60846820       5.416667    London
## 8  United Kingdom 2007 50134.32     61322463       5.333333    London
## 9  United Kingdom 2008 46767.59     61806995       5.708333    London
## 10 United Kingdom 2009 38262.18     62276270       7.608333    London
## 11 United Kingdom 2010 38893.02     62766365       7.891667    London
## 12 United Kingdom 2011 41412.35     63258918       8.091667    London
## 13 United Kingdom 2012 41790.78     63700300       7.991667    London
## 14 United Kingdom 2013 42724.07     64128226       7.591667    London
## 15 United Kingdom 2014 46783.47     64613160       6.200000    London
## 16 United Kingdom 2015 44305.55     65128861       5.375000    London
## 17 United Kingdom 2016 40341.41     65637239       4.900000    London
##    country.y    gdp.y population.y unemployment.y capital.y
## 1    Ireland 26241.51      3805174       4.266667    Dublin
## 2    Ireland 28227.28      3866243       3.925000    Dublin
## 3    Ireland 32539.95      3931947       4.491667    Dublin
## 4    Ireland 41107.03      3996521       4.616667    Dublin
## 5    Ireland 47630.93      4070262       4.491667    Dublin
## 6    Ireland 50878.64      4159914       4.400000    Dublin
## 7    Ireland 54306.91      4273591       4.525000    Dublin
## 8    Ireland 61359.64      4398942       4.691667    Dublin
## 9    Ireland 61257.90      4489544       6.433333    Dublin
## 10   Ireland 52104.04      4535375      12.050000    Dublin
## 11   Ireland 48671.89      4560155      13.908333    Dublin
## 12   Ireland 52224.01      4576794      14.675000    Dublin
## 13   Ireland 49177.44      4586897      14.725000    Dublin
## 14   Ireland 52060.47      4598294      13.091667    Dublin
## 15   Ireland 55899.16      4617225      11.316667    Dublin
## 16   Ireland 62139.67      4676835       9.458333    Dublin
## 17   Ireland 63861.92      4773095       7.908333    Dublin
```

Notice that instead of duplicated names for the columns, we now have a suffix .x and .y. It might be useful to have a more informative suffix so that we don’t have to constantly remember which country is x and which country is y.

```r
full_join(UK,Ireland,by="date",suffix=c("_UK","_IRE"))
```

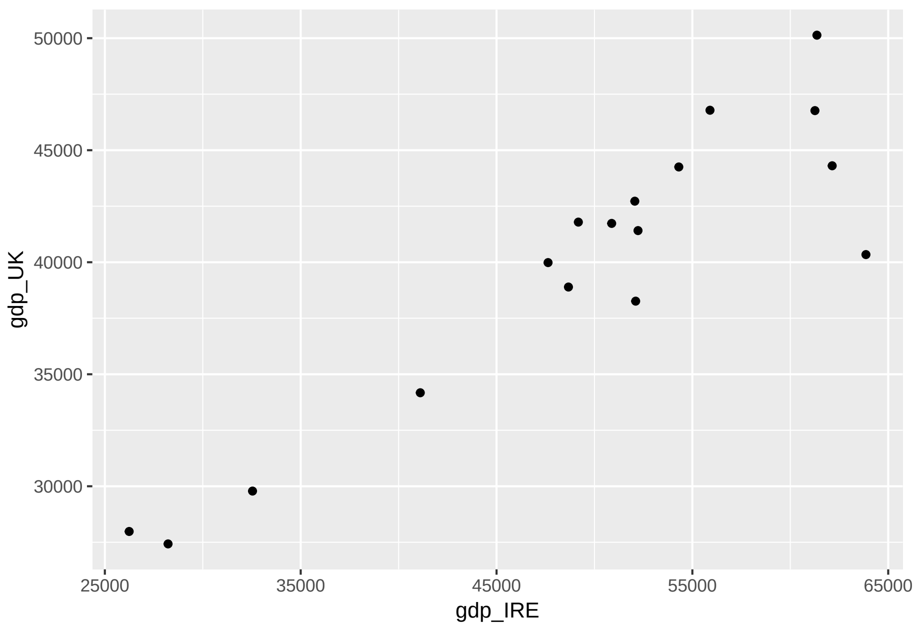
QUESTION: In the previous chunk what we would expect to see if we were to replace full_join() with inner_join()?

7.4 Applying merging

7.4.1 Piping into ggplot (+ some new geoms)

We could now make a plot of GDP in Ireland against GDP in UK. We can pipe data into a graph in a similar way to piping data commands that we saw yesterday.

```r
full_join(UK,Ireland,by="date",suffix=c("_UK","_IRE")) %>%
  ggplot(aes(y=gdp_UK,x=gdp_IRE)) +
    geom_point()
```

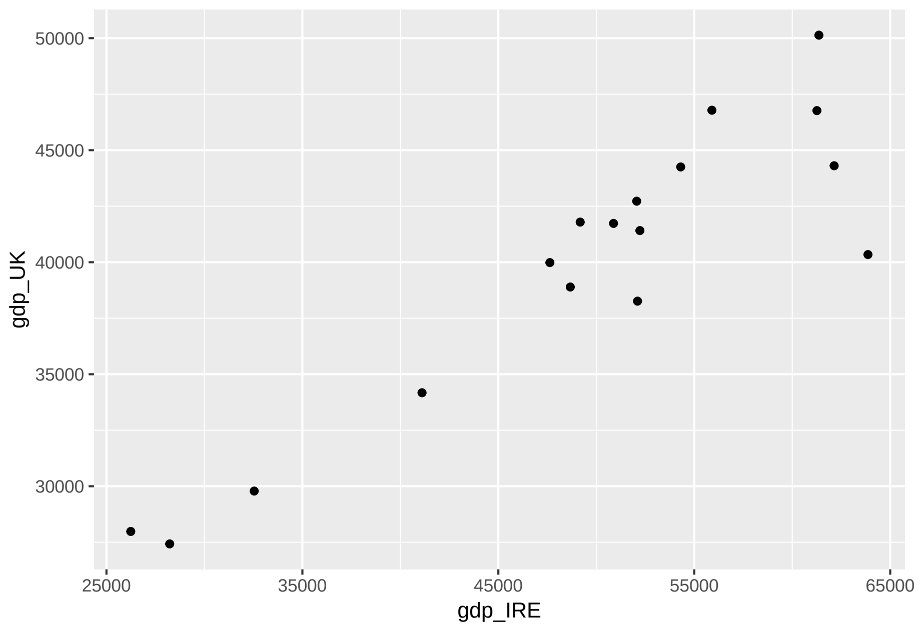


Remember that once we have started the ggplot() we use the + to link together the components of the graph. The %>% is used to make modifications to the data.

We can also assign our plots to be named objects, similar to how we have named data frames so far. This can be a good way of sequentially building up a plot. But to actually see the plot we need to repeat back the name.

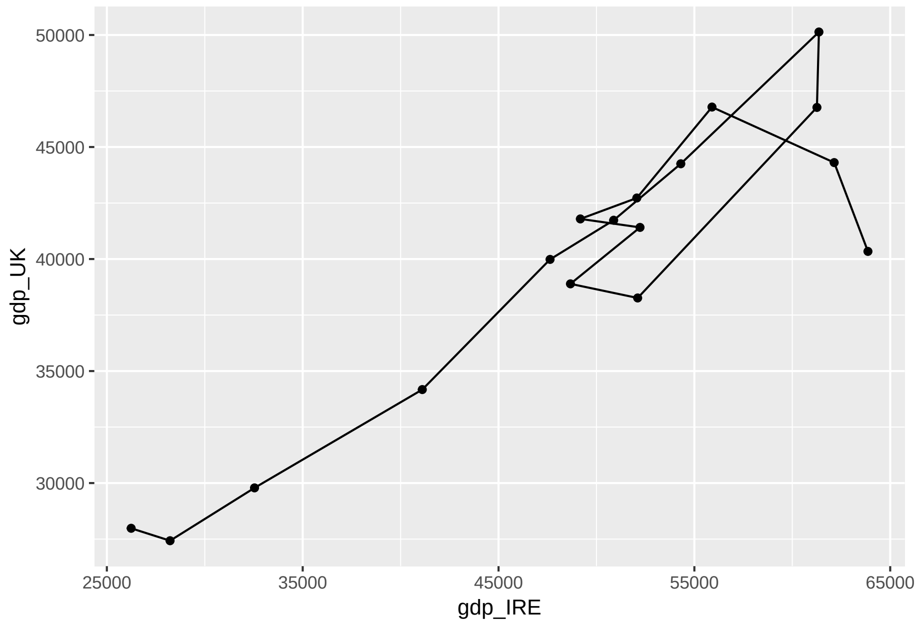
```r
plot1<-full_join(UK,Ireland,by="date",suffix=c("_UK","_IRE")) %>%
  ggplot(aes(y=gdp_UK,x=gdp_IRE)) +
    geom_point()

plot1
```



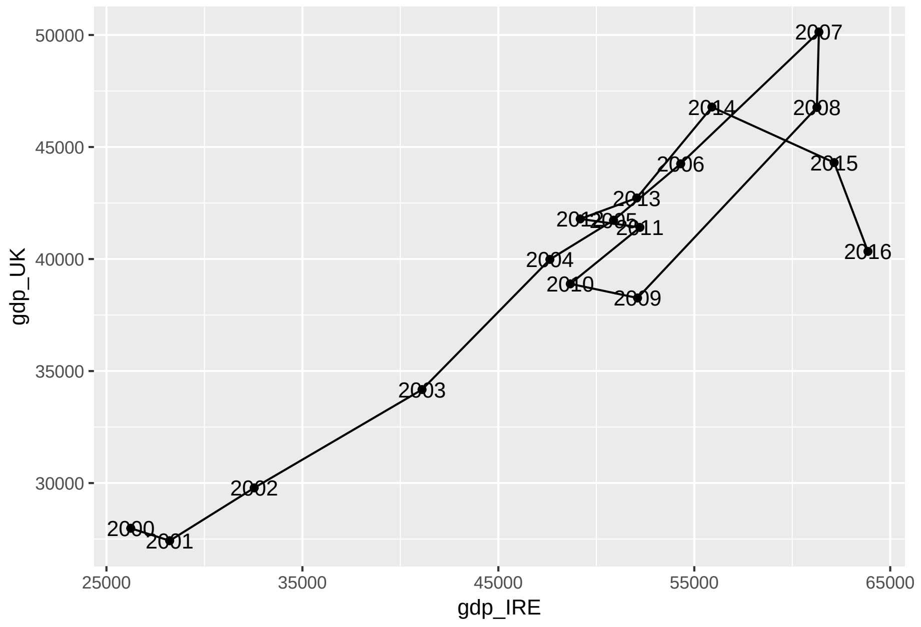
Maybe we want to show the time dimension on this graph in some way, as each point represents a year. Yesterday we used geom_line() to make line graphs but in this case it may not be so useful. geom_line joins up the points from left to right. But that may not be the same order as it is from year to year. So instead we can use the function geom_path, which joins the points in data order. If we want to be sure that the data order is sensible, we can make sure we arrange it by date first.

```r
plot1+
  geom_path()
```



To show this even clearer we could label the points. geom_text() adds text to the graph. and the text column needs to be assigned to an extra aesthetic called "label".

```
plot1+
  geom_path()+
    geom_text(aes(label=date))
```



QUESTION:
Modify the previous code to produce the same plot but; 1. colour the text labels red 2. place the labels above the points instead of on top of the points; 3. make the points bigger; 4. use a dotted line instead of a full line. 5. You might need to look into the help menus for some of these geoms to find the appropriate options, or remember the (probably even more useful) R Graphics Cookbook: http://www.cookbook-r.com/Graphs/

```
plot1+
  ???
```

Another way of helping identify which point is which would be to use an interactive graph using the `plotly` library. This is a really easy way of making a really nice interactive graph.

To use plotly you need to assign your plot to an object, and then run ggplotly() around that object. Any information you want to be available in your interactive plot needs to be mapped to an aesthetic in the ggplot statement.

```
ggplotly(plot1)
```

But these don't have to be real geoms, you can use any name and then this will be carried through to the interactive plot. E.g. - this code (suprisingly) works:

```
plot2<-full_join(UK,Ireland,by="date",suffix=c("_UK","_IRE")) %>%
  arrange(date) %>%
   ggplot(aes(y=gdp_UK,x=gdp_IRE,sgndskjgfbsljk=date)) +
    geom_point()+
      geom_path()

ggplotly(plot2)
```

7.4.2 Challenging Merging

Let's bring in another dataset, from a different source, to do some more interesting, and difficult, merging tasks.

The Livestock data contains data from a similar period, for all countries in the world, on the number of different livestock units within that country.

Click on the dataset in the environment window to take a look at the data.

Let's try and merge this with the combined data from UK & Ireland (UKIreland).

QUESTION: Take a close look at these two datasets. Can you see any possible issues with trying to merge these datasets together?

These issues can be resolved by having i) multiple key join fields, brought together using c() ii) changing the specification slightly - insteady of by="key__column__name" we can have `by=c("key_column_name_from_1st_data"="key_column_name_f`

Before running the code try to answer the following question:

QUESTION: How many rows and columns would we expect to see in the result of each of these join statement. HINT: Check the Environment tab for lthe size of the data frames.

```
full_join(UKIreland,Livestock,by=c("country"="Area","date"="Year"))

inner_join(UKIreland,Livestock,by=c("country"="Area","date"="Year"))

anti_join(UKIreland,Livestock,by=c("country"="Area","date"="Year"))

anti_join(Livestock,UKIreland,by=c("Area"="country","Year"="date"))
```

QUESTION: We are interested in comparing the livestock populations of Ireland and the UK. Which of these options would be the most useful of these joins to use for further analysis? Assign the most useful join to an object called UKIrelandLivestock

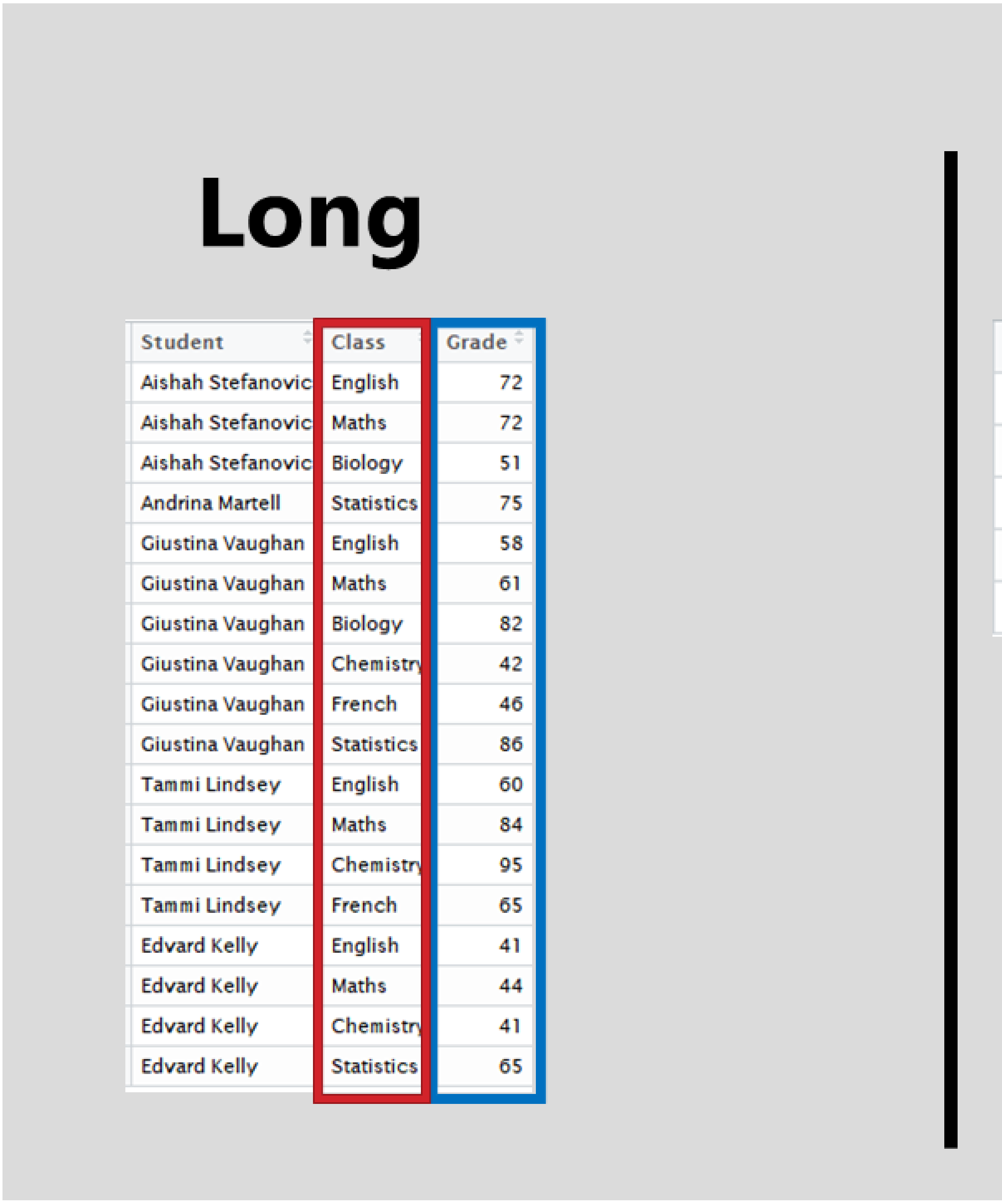
```
UKIrelandLivestock<-
```

Chapter 8

Reshaping Data

Let's say you and a colleauge are asked to support data analysis of a school. You are asked to look at the grades of five students. For each student you need to look at the their grades in different subjects.

You both go away, collect your data and come back to share your results. The image below are the tables of your result. They may look different but these two datasets are showing the same information. They different only in shape; one is wide and the other is long.

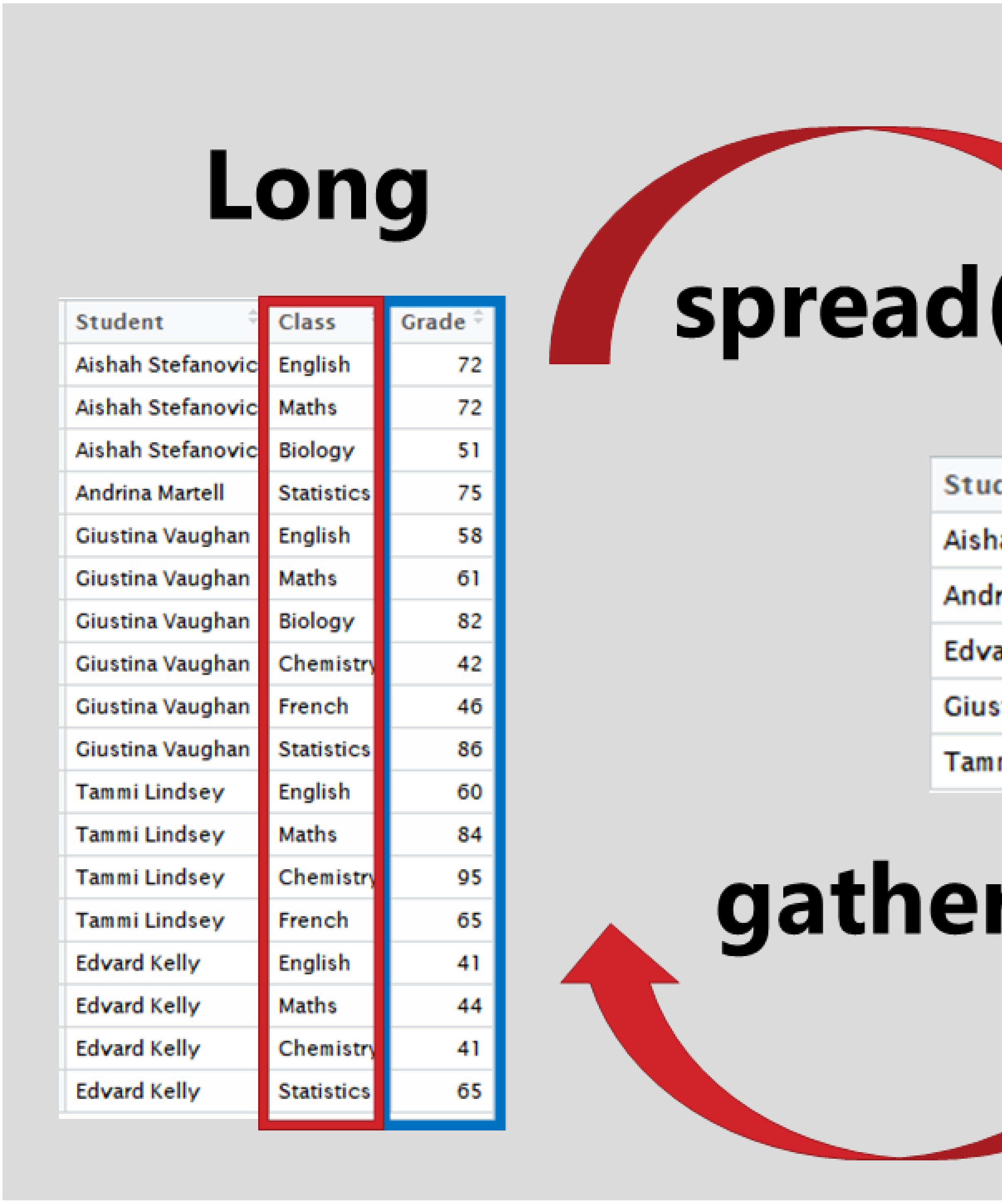


Long

Student	Class	Grade
Aishah Stefanovic	English	72
Aishah Stefanovic	Maths	72
Aishah Stefanovic	Biology	51
Andrina Martell	Statistics	75
Giustina Vaughan	English	58
Giustina Vaughan	Maths	61
Giustina Vaughan	Biology	82
Giustina Vaughan	Chemistry	42
Giustina Vaughan	French	46
Giustina Vaughan	Statistics	86
Tammi Lindsey	English	60
Tammi Lindsey	Maths	84
Tammi Lindsey	Chemistry	95
Tammi Lindsey	French	65
Edvard Kelly	English	41
Edvard Kelly	Maths	44
Edvard Kelly	Chemistry	41
Edvard Kelly	Statistics	65

Long data sets are preferrable when working on the data on a computer whereas wide formatted data works best when presenting this information. It is unlikely the problem you will be working will provide you with perfectly formatted datasets, so it is helpful to know how to move between these shapes.

`tidyr` is a R package that aims to solve this problem by adding two useful "words" to our vocabulary `gather()` and `spread()`.



Student	Class	Grade
Aishah Stefanovic	English	72
Aishah Stefanovic	Maths	72
Aishah Stefanovic	Biology	51
Andrina Martell	Statistics	75
Giustina Vaughan	English	58
Giustina Vaughan	Maths	61
Giustina Vaughan	Biology	82
Giustina Vaughan	Chemistry	42
Giustina Vaughan	French	46
Giustina Vaughan	Statistics	86
Tammi Lindsey	English	60
Tammi Lindsey	Maths	84
Tammi Lindsey	Chemistry	95
Tammi Lindsey	French	65
Edvard Kelly	English	41
Edvard Kelly	Maths	44
Edvard Kelly	Chemistry	41
Edvard Kelly	Statistics	65

For these functions, you need to identify two things; **key** and **value**. **key** is the identifier and **value** are the measurements. In the example, the **key** identifier is `Class`, highlighted in red and **value** are the `Grades`, highlighted in blue.

8.1 Going Long

Let's try to apply these concepts to the the dataset we merged together in the last tutorial.

After loading the UK and Ireland data sets, run the code below to merge the datasets into one.

```
UKIrelandLivestock<-inner_join(UKIreland,Livestock,by=c("country"="Area","date"="Year"))
```

```
## Warning: Column `country`/`Area` joining character vector and factor,
## coercing into character vector
```

Our task is to adjust the number of livestock in each category for the country population. This might provide a fairer comparison between the UK and Ireland over time.

QUESTION: Write a line of code to calculate a column containing the number of Turkeys per capita in the merged dataset

Currently our livestock information is in wide format: one column for each type of livestock containing the values. For our data processing it would be more efficient to work with data in long format: two columns - the first containing an identifier for livestock type and the second containing the values.

We could apply a similar function on every single livestock column in our data. This would be quite tedious! Or we could `gather()` our livestock columns together and then divide one single column by the population.

There are 3 arguments which are required for `gather()`;

- The first is the new name for the column which will contain the **key**. This will take the values that are currently assigned to the column names.
- The second is the new name for the column which wil contain the **values**, which will contain the data values from the columns.
- The third is to identify which columns to include. If these columns are sequential we can use a shortcut by using a colon and saying: FirstColumn:LastColumn

An option to use in our function is `na.rm=TRUE`. This will remove any missing values from our data. There are no camels as livestock in the British Isles, so adding this option will remove the rows about camels from the data

```
UKIrelandLivestock %>%
  gather("livestock","total",Beehives:Turkeys,na.rm=TRUE)
```

The same could also have been achived by fully specifying each of the columns to be gathered, and only choosing the animals relevant to the British Isles

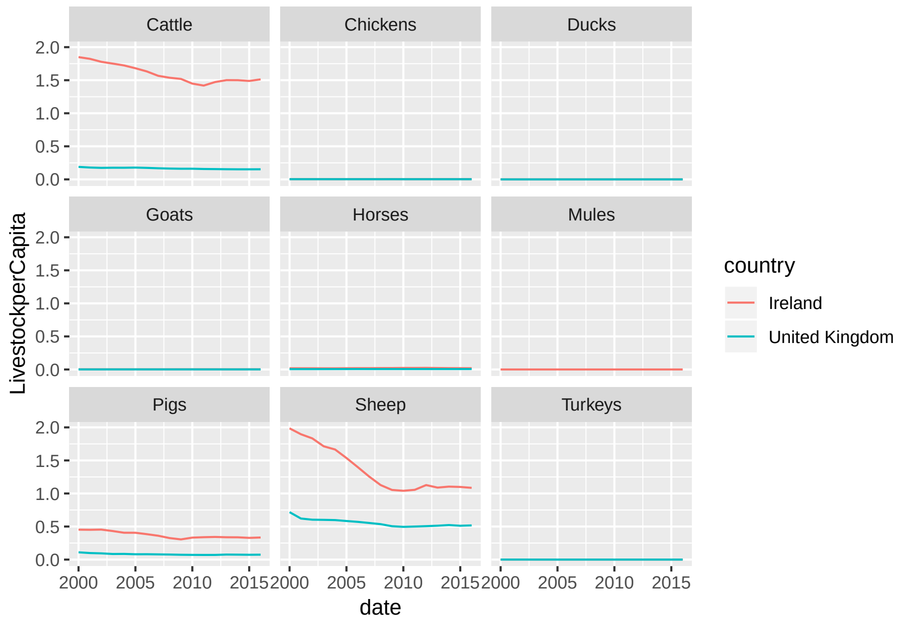
```
UKIrelandLivestock %>%
  gather("livestock","total",Cattle,Chickens,Ducks,Goats,Horses,Pigs,Sheep,Turkeys)
```

QUESTIONS: Extend the previous line to add in a column of livestock per capita using the mutate function and assign it to a new object called UKIrelandLivestockCap

```
UKIrelandLivestockCap<-
  UKIrelandLivestock %>%
  gather("livestock","total",Beehives:Turkeys,na.rm=TRUE) %>%
  ?????
```

Gathering multiple columns is also a really useful way to be able to plot multiple variables in the same ggplot.

```
UKIrelandLivestockCap %>%
    ggplot(aes(y=LivestockperCapita,x=date,colour=country))+
        geom_line()+
          facet_wrap(~livestock)
```

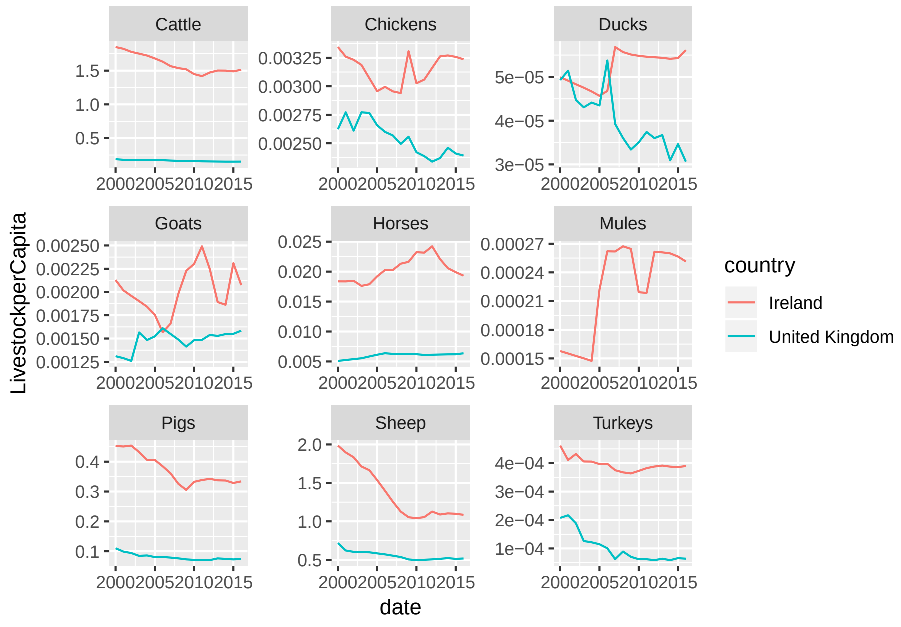


Did this work for you? If not why not? - what needs to be changed?

This graph is a little confusing right now. Too many of the animals have very small numbers so are effectively invisible.

So let's make it better - we can set our facets to have different scales using the option scales="free"

```
UKIrelandLivestockCap %>%
    ggplot(aes(y=LivestockperCapita,x=date,colour=country))+
        geom_line()+
          facet_wrap(~livestock,scales="free")
```



One thing you probably noticed (other than the fact that the sheep population is declining at an alarming rate), is that this graph is a little bit squashed in at the moment within the RMD file. You can press the small square button in the top right corner of the plot to open a new window to view, and re-size the graph. It's best to use a little bit of trial and error to get a graph the size you want it to be.

8.2 Going wide

When presenting results, we usually want to put the data back into a wide format as this is easier for humans to read.

Let's say we would like to present the cattle per capita figures for UK and Ireland over the time period.

Spread requires two arguments.

The first is the column containing the identifying information. Each unique value in this column will be a column name in our wide data

The second is the column containing the corresponding values. These will form the data within each column in the wide data.

```
UKIrelandLivestockCap %>%
     filter(livestock=="Cattle") %>%
       select(date,country,LivestockperCapita) %>%
         spread(country,LivestockperCapita)
```

QUESTION: What happens if you do not include the select() line in the code? Can you explain why?

```
UKIrelandLivestockCap %>%
     filter(livestock=="Cattle") %>%
         spread(country,LivestockperCapita)
```

8.2.1 Exercises

QUESTION Read in the data frame used in the lecture notes with the information from a school database of classes, grades and students. This can be found at: http://shiny.stats4sd.org/Reading_R/Class Data.xlsx or in the data folder in RStudio Cloud

```
Grades<-
Classes<-
Students<-
```

QUESTION: Using the example from the lecture notes summarising the average grade in each class as a starting point (reproduced below): i) Produce a summary of the average grade achieved by each student; ii) The number of classes that student attended; iii) Then merge this data with the Students data frame containing the student-level information

```
Grades %>%
  group_by(Class) %>%
    summarise(GradeAverage=mean(Grade) ,Students=n()) %>%
        full_join(Classes,by="Class")
```

QUESTION: From the Livestock dataset produce a table with one column for each year showing the total number of Chickens for France

```
Livestock %>%
  filter() %>%
    select() %>%
      spread()
```

QUESTION: From the UKIreland data gather the columns gdp, population and unemployment to be able to produce a facetted plot showing all of these variables over time for the UK and Ireland

```
UKIreland %>%
  gather() %>%
    ggplot(aes()) +
      facet_wrap()+
        geom_XXXX()
```

Chapter 9

Modelling

9.1 Set Up for Session

Packages used = ggplot2, ciTools, dplyr, ggfortify Data used: eastbourne-data.csv

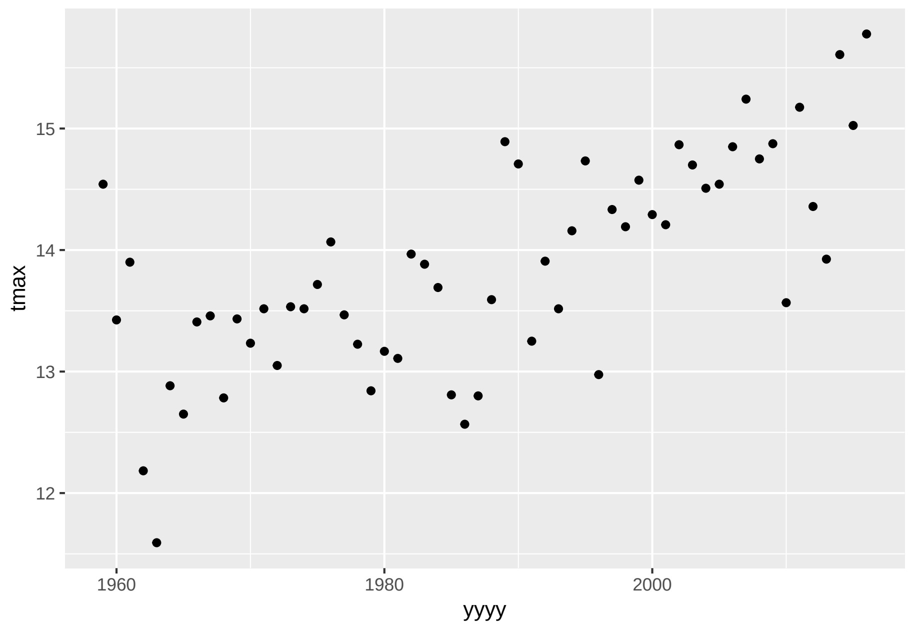
```
library(dplyr)
library(ggplot2)
library(ciTools)
library(ggfortify)

eastbourne<-read.csv("C:/Users/FaheemAshraf/Dropbox (SSD)/Faheem_workings/2018-11-22 - R Course E
```

9.2 Exploring Data

The data for this example is taken from the UK Met office and contains the average daily maximum temperatures, minimum temperatures and rainfall for all years since 1959.

```
plot1<-ggplot(data=eastbourne,aes(y=tmax,x=yyyy))+
  geom_point()
plot1
```

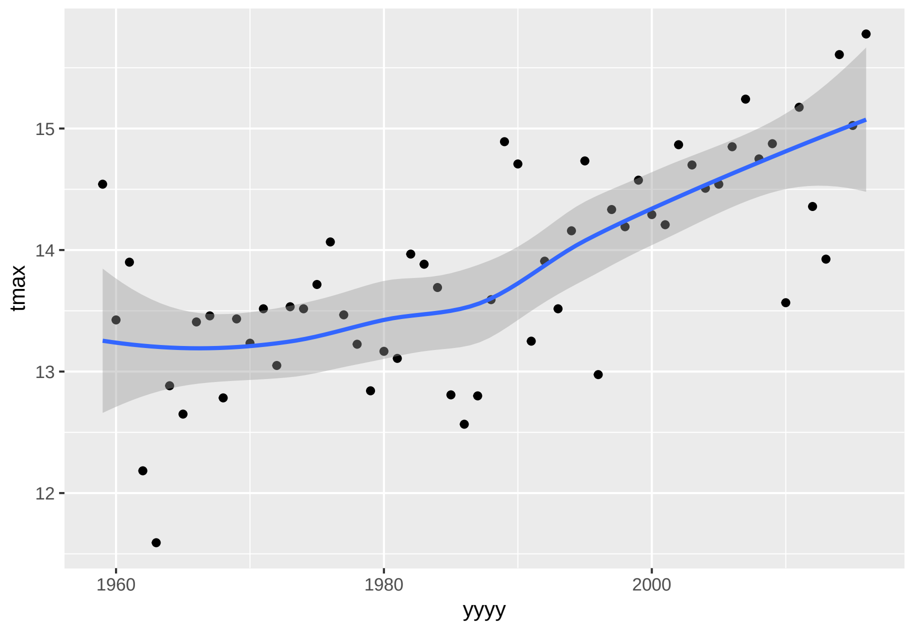


What would be our initial thoughts about the relationship between temperature and time in Eastbourne?

`geom_smooth` is a nice way to add on simple models to your plot that we have seen before.

```
plot1 +
    geom_smooth()
```

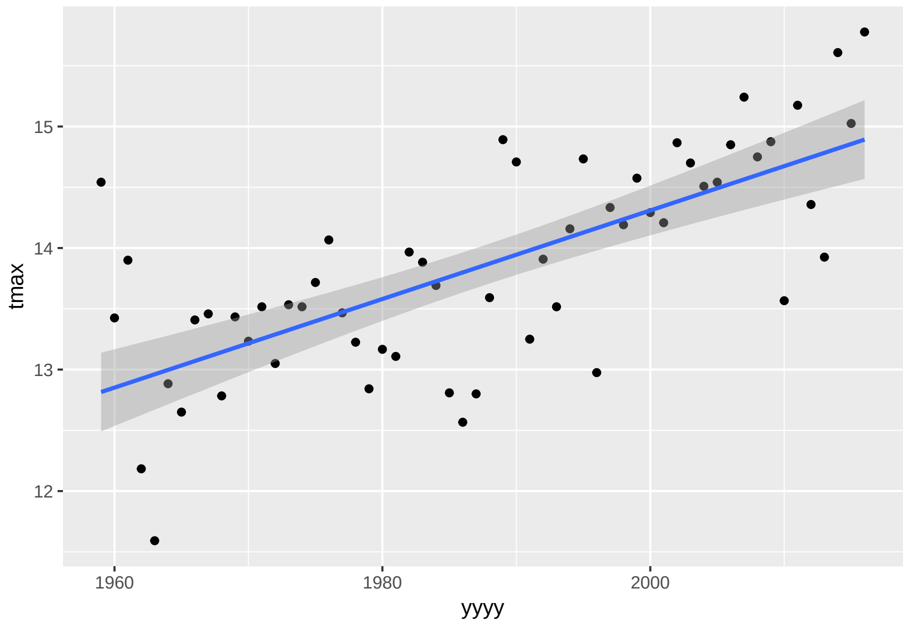
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



The default model fitted by geom_smooth depends on the amount of data being plotted. In this case it has told us that it is using a "loess" model to fit the line - this is a simple form of a moving average smoother. It can be useful for visualising trends that do not fit into other models, but they cannot be used for extensive statistical inference.

Lets use a more familiar model by changing the "method" option into a simple linear regression:

```
plot1 +
    geom_smooth(method="lm")
```



9.3 Fitting Model

To examine what we actually have in our model we need to fit the model using the lm() command:

```
lm(tmax~yyyy,data=eastbourne)
```

```
##
## Call:
## lm(formula = tmax ~ yyyy, data = eastbourne)
##
## Coefficients:
## (Intercept)         yyyy
##   -58.58684      0.03645
```

The output by default only tells us two things: “Call” - simply repeating back the model we have specified “Coefficients”: Telling us the values of the parameters

A linear regression follows the equation of a straight line y = B0 + B1x (or y=a+bx or y=mx+c ; depending on where and when you were 12 years old) The coefficients give us the value of our intercept: -58.6 and the value of our slope: 0.036. So the overall model would be:

tmax = -58.6 + 0.036*yyyy

This means if yyyy=0 we would expect the maximum temperature in that year to be -58.6 degrees (!!!) and for every one year increase the average temperature increase is 0.036 degrees.

To get more output then we need to save the model to an object:

```
tmaxmodel<-lm(tmax~yyyy,data=eastbourne)
```

Then there are lots of functions that give us different pieces of output and inference from this model.

```
summary(tmaxmodel)
```

```
##
## Call:
## lm(formula = tmax ~ yyyy, data = eastbourne)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.3691 -0.3777  0.0604  0.3211  1.7267
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -58.586839   9.724527  -6.025 1.39e-07 ***
## yyyy          0.036448   0.004893   7.450 6.25e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6238 on 56 degrees of freedom
## Multiple R-squared:  0.4977, Adjusted R-squared:  0.4888
## F-statistic:  55.5 on 1 and 56 DF,  p-value: 6.255e-10
```

summary(model) provides us with a lot of useful information - model fit statistics (R squared values & F statistic), standard errors and p-values for the coefficients.

One thing missing is confidence intervals, this comes from confint:

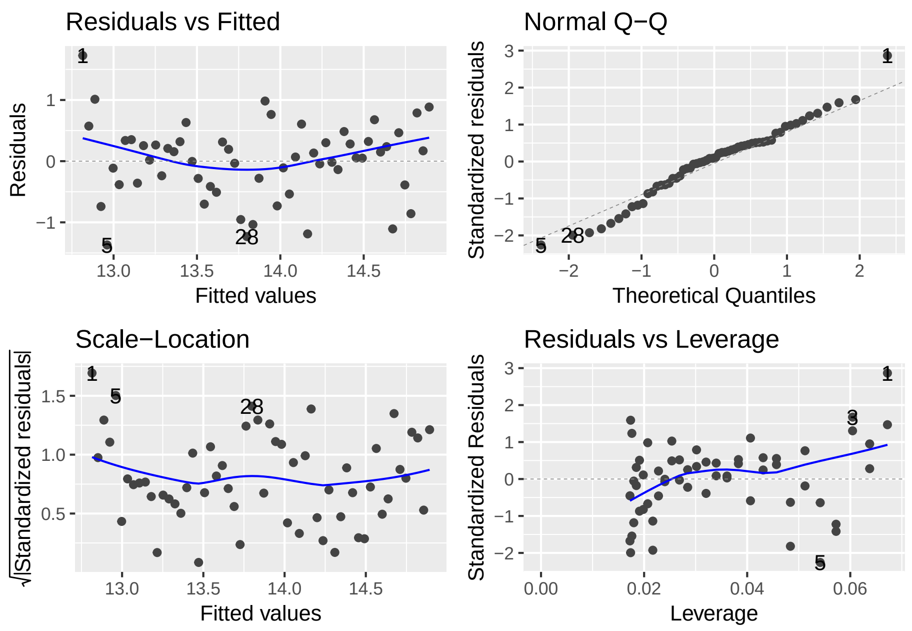
```
confint(tmaxmodel)
```

```
##                    2.5 %       97.5 %
## (Intercept) -78.06740617 -39.10627121
## yyyy          0.02664692   0.04624931
```

9.4 Checking Model

We should also check our model fit plots to assess model validity. `autoplot(model)` from the ggfortify package produces 4 model checking plots. When we are working with time series data it is also very useful to produce an extra plot - the acf (auto-correlation plot) to assess the strength of temporal correlation.

```r
autoplot(tmaxmodel)
```



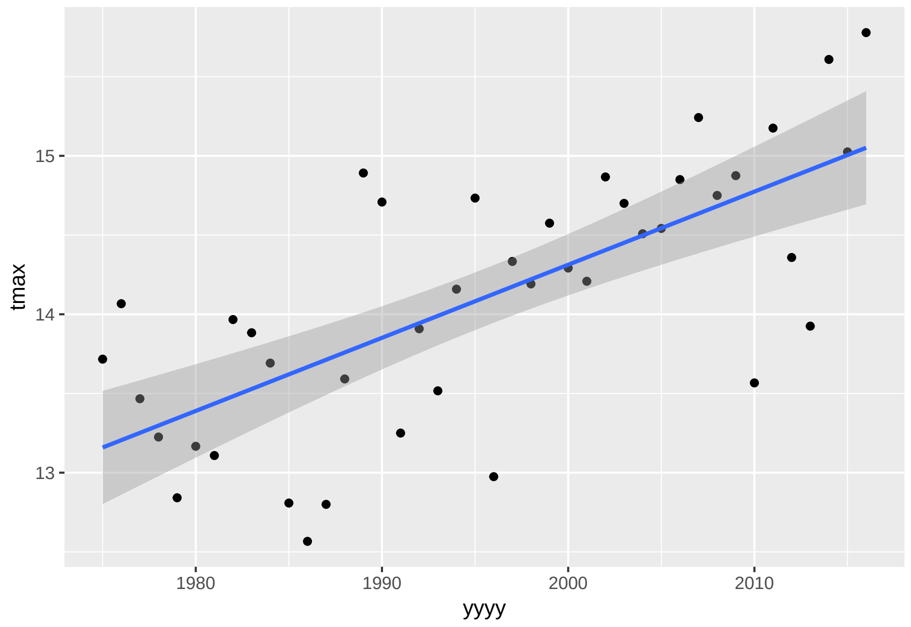
Do these plots look OK? What are we actually looking for here?

In particular we can see a problem with the trend in the residual vs fitted plot as there is clear curvature in the pattern. Does this make sense if we go back to our original scatter plots and the loess models fitted by geom_smooth

9.5 Updating Model

One solution to solve the problem we have might be to fit a model only on the years between 1975 to 2015, since the trend during this range appears to be relatively linear. We can filter our data and then pipe through to lm:

```
eastbourne %>%
  filter(yyyy>=1975) %>%
    ggplot(aes(y=tmax,x=yyyy))+
     geom_point()+
       geom_smooth(method="lm")
```



```
eastbourne %>%
  filter(yyyy>=1975) %>%
    lm(tmax~yyyy,data=.)
```

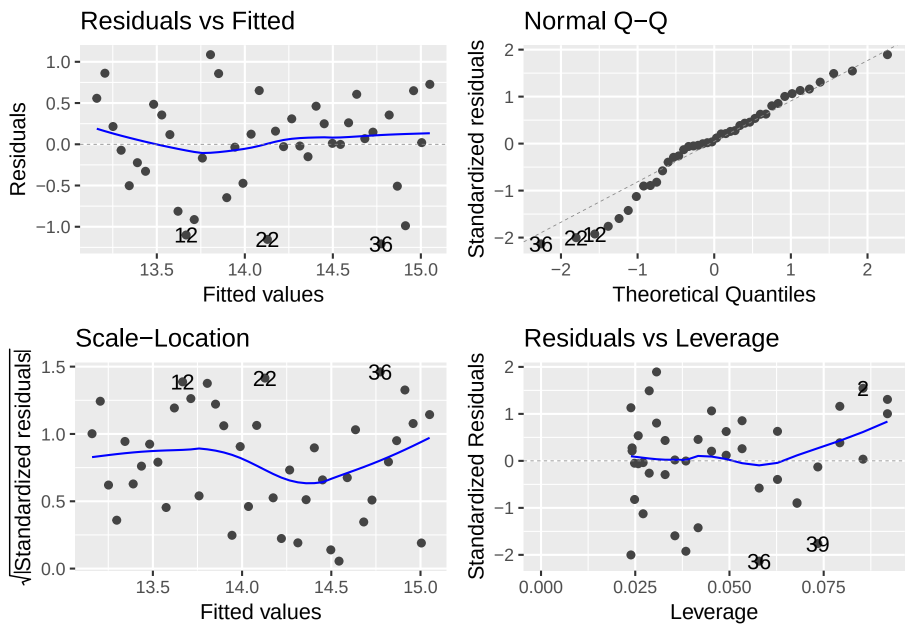
```
##
## Call:
## lm(formula = tmax ~ yyyy, data = .)
##
## Coefficients:
## (Intercept)         yyyy
##   -77.97949      0.04615
```

This is slightly different to what we have seen previously with pipes because the first argument of lm() is the formula and not the data. Using the pipe automatically puts the data into the first argument of the next line. For lm(), and other functions where data is not the first argument, then saying "data=." tells R where to put the data from the previous line.

Again we need to assign this to an object to do other useful things! Let's first check the model validity plots to see if they look any better:

```
tmaxmodel_1975<-
  eastbourne %>%
  filter(yyyy>=1975) %>%
    lm(tmax~yyyy,data=.)

autoplot(tmaxmodel_1975)
```



This looks better. Let's take another look at these result:

```
summary(tmaxmodel_1975)
```

```
##
## Call:
## lm(formula = tmax ~ yyyy, data = .)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.20748 -0.30145  0.04414  0.35503  1.08659
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -77.979495  14.810825  -5.265 5.07e-06 ***
## yyyy          0.046146   0.007422   6.217 2.34e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.583 on 40 degrees of freedom
## Multiple R-squared:  0.4915, Adjusted R-squared:  0.4788
## F-statistic: 38.66 on 1 and 40 DF,  p-value: 2.337e-07
```

9.6 Confidence and Prediction Intervals

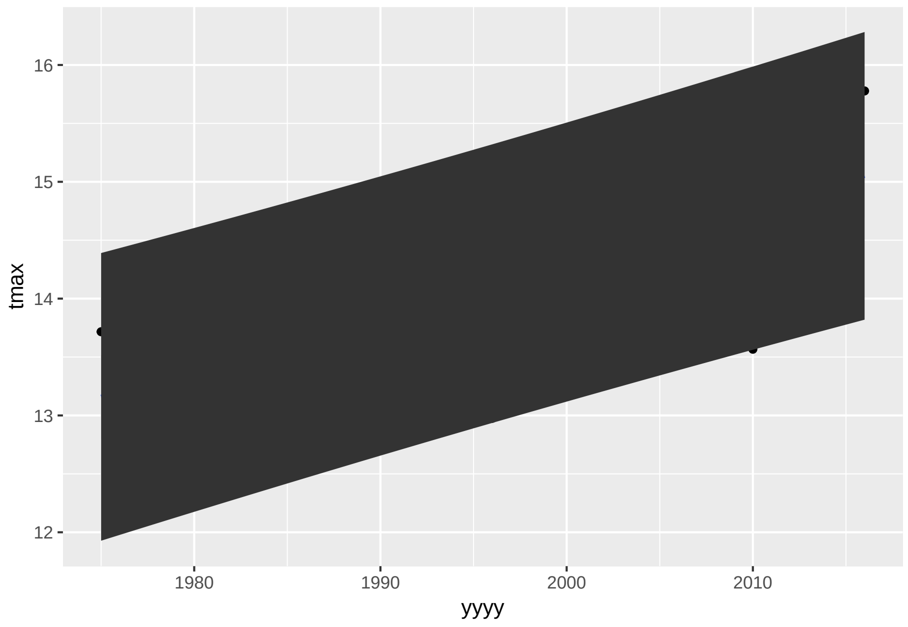
The geom_smooth function fits confidence intervals around the line. This tells us the margin of error in the expected value of temperature in each year. This is OK, but it can also be very useful to include prediction intervals. This tells us the range where we would expect any individual observation to be.

The functions add_ci and add_pi can add in confidence intervals and prediction intervals to our original data. These work nicely in pipes, where we simply need the name of the model inside these functions.

```
eastbourne_int<-eastbourne %>%
  filter(yyyy>=1975) %>%
    add_pi(tmaxmodel_1975) %>%
      add_ci(tmaxmodel_1975)
```

To add in errors as lines this needs the `geom_ribbon` geom from ggplot2.

```
ggplot(eastbourne_int,aes(x=yyyy,y=tmax))+
 geom_point() +
  geom_smooth(method="lm")+
   geom_ribbon(aes(ymax=UPB0.975,ymin= LPB0.025))
```



QUESTION - This graph is 'correct', but looks horrible. Can you find the options to make this plot look a bit better? Instead of a big black blob, how can we have something more similar to the way the error bar gets plotted in geom_smooth() - transparent and shaded in a colour rather than black?

We can also use add_ci or add_pi to give us future predictions to extrapolate our model into the future

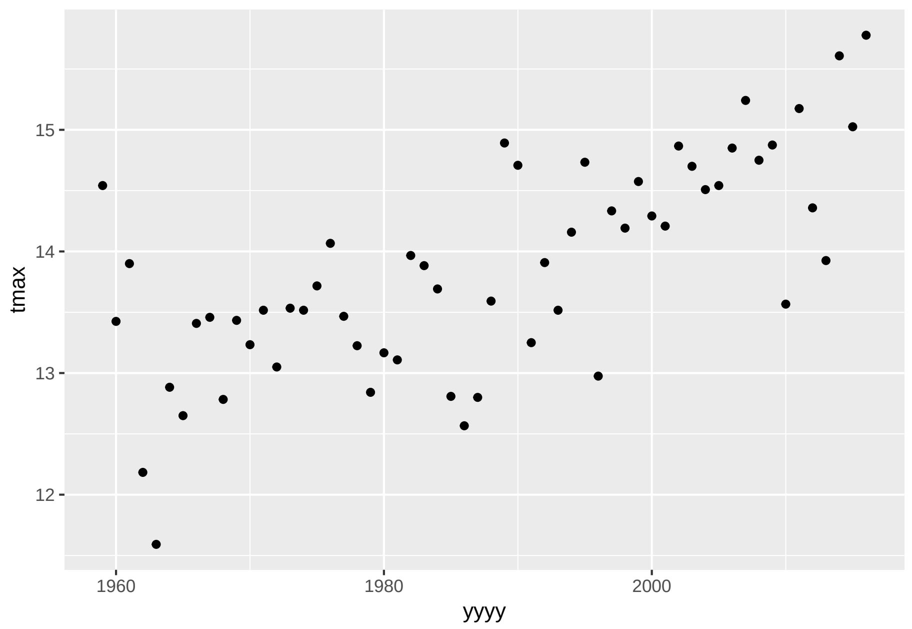
```
predictiondata<-data.frame(yyyy=1975:2100) %>%
   add_ci(tmaxmodel_1975) %>%
     add_pi(tmaxmodel_1975)
predictiondata
```

```
## # A tibble: 126 x 6
##      yyyy  pred LCB0.025 UCB0.975 LPB0.025 UPB0.975
##     <int> <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
##  1  1975  13.2     12.8     13.5     11.9     14.4
##  2  1976  13.2     12.9     13.5     12.0     14.4
##  3  1977  13.3     12.9     13.6     12.0     14.5
##  4  1978  13.3     13.0     13.6     12.1     14.5
##  5  1979  13.3     13.0     13.7     12.1     14.6
##  6  1980  13.4     13.1     13.7     12.2     14.6
##  7  1981  13.4     13.2     13.7     12.2     14.6
##  8  1982  13.5     13.2     13.8     12.3     14.7
##  9  1983  13.5     13.3     13.8     12.3     14.7
## 10  1984  13.6     13.3     13.8     12.4     14.8
## # ... with 116 more rows
```

We can plot our predicted model into the future onto a nice graph as well. But - we have our raw data in one file and our predicted data in another file. So we are going to show you how to use ggplot to make a single plot from multple data frames.

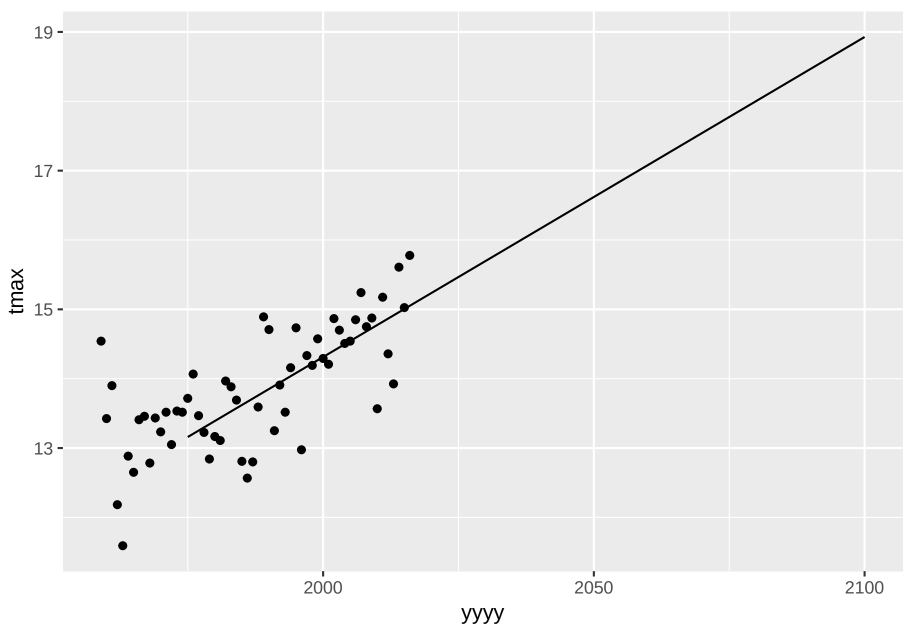
Using plot1 (saved earlier) - which had used the data=eastbourne.

```
plot1
```



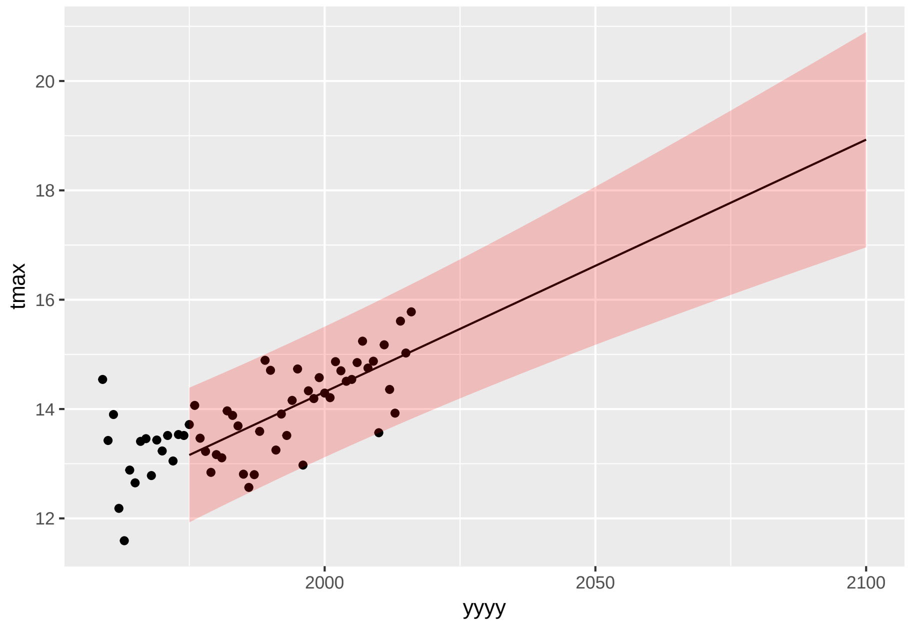
We can add in the aesthetics and data into any geom. But we need to add in also an option inherit.aes=FALSE. This tells R to ignore the 'global' aesthetics which were set in the original ggplot() statement.

```
plot1+
  geom_line(aes(x=yyyy,y=pred),data=predictiondata,inherit.aes = FALSE)
```



We can then add our prediction interval in the same way:

```
plot1+
  geom_line(aes(x=yyyy,y=pred),data=predictiondata,inherit.aes = FALSE) +
    geom_ribbon(aes(x=yyyy,ymax=UPB0.975,ymin=LPB0.025),data=predictiondata,alpha=0.2,i
```



9.7 Extending to multiple regression:

We can add variables into our model using +. Let's add rainfall as a potential predictor for tmax.

```
tmaxrain<-lm(tmax~yyyy+rain,data=eastbourne)
summary(tmaxrain)
```

```
##
## Call:
## lm(formula = tmax ~ yyyy + rain, data = eastbourne)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.36728 -0.37668  0.06186  0.31932  1.72483
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.850e+01  9.889e+00  -5.916 2.20e-07 ***
## yyyy         3.642e-02  4.950e-03   7.359 9.74e-10 ***
## rain        -5.395e-04  7.796e-03  -0.069    0.945
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 0.6294 on 55 degrees of freedom
## Multiple R-squared:  0.4978, Adjusted R-squared:  0.4795
## F-statistic: 27.26 on 2 and 55 DF,  p-value: 5.95e-09
```

It looks like there is no evidence of a relationship between rainfall and maximum temperatures.

9.8 Exercise

Repeat the whole modelling process for the column tmin instead of tmax. Start by making a scatter plot, using geom_smooth to fit a linear model onto this graph and then use lm() to fit a model, check the validity, draw conclusions about how tmin has changed over time, if so by how much, and make predictions for the next 5 years

- Start with producing a scatter plot, using geom_smooth to fit lines, using booth the smoother and the linear regression

```
ggplot(data=eastbourne,aes(y=??,x=???))+???
```

- Then fit a model of minimum temperature against year:
- Then check the model validity:
- Then look at and interpret the summary table:
- Then make a plot showing the fitted model, and add some prediction intervals around your model:
- (Difficult question) Make a plot showing both tmin and tmax on the same axes and the fitted models for both.

Chapter 10

Simulation

10.1 Description of the session goal

In this session we will learn about: *Packages*
Sampling
Functions
Loops
At the end we'll try to make a concrete use of all of it and answer an interesting question in probability.

Ready?

10.2 Find and install R packages

A nice website to find packages is https://www.rdocumentation.org
That's how I found about the package "ukbabynames" that contains data for all baby names given at least three times between 1996 and 2015.

To install a package called "ukbabynames", we do

```r
install.packages("ukbabynames")
```

Once a package is installed in your computer, it will remain installed and so you only need to install the package again if you're updating your version of R.

However, each time you open a new session of R or Rstudio, you need to load the package if you want to use it:

```
library(ukbabynames)
```

I'm sure you noticed that we've done it several times already with packages like `ggplot2` or `dplyr()`.

Most often we are interested in the functions of the packages we install, but a package can also contain datasets and in this case, the package "ukbabynames" contains a dataframe of the same name and this is what we will use from it:

```
View(ukbabynames)
```

QUESTION 2.1: The function `say()` from the package "cowsay" is useless, but funny. Install the package "cowsay". Then, when the installation is finished (you have a to wait a bit, like for any package you would install), load the package and try to run the command `say(Hello R!)`. What happens?

```
???.???("cowsay")
```

```
???(cowsay)
say("Hello R!")
```

10.3 Sampling from data or known distributions

10.4 Sampling from data

The `sample()` function allows you to choose a random sample from a list of values.

```
sample(1:30, 5)
```

```
## [1] 18  9 29 17  1
```

The first argument is the values to sample from and the second is the number of values to sample. If we omit the second argument, we get all the values, but in a random order:

```
sample(1:12)
```

```
##  [1]  7  3 10  8  2  4 12  9  5 11  6  1
```

By default `sample()` samples without replacement, so when you do

```
sample(1:5, 10)
```

You get an error. To sample with replacement you need to add the argument replace=TRUE:

```
sample(1:5, 10, replace=TRUE)
```

```
##  [1] 3 5 3 5 2 3 4 5 5 2
```

10.5 Generating random numbers from distributions

We can also generate random numbers from known distribution.

runif (uniform)
rpois (poisson)
rnorm (normal)
rbinom (binomial)
rgamma (gamma)
rbeta (beta)

The first argument for all of these is n, the number of samples to generate. Following arguments can specify parameters for the distribution.

Here are three random numbers coming from a standard normal distribution:

```
rnorm(3)
```

```
## [1]  0.1005724 -0.4920847  0.1394214
```

here, six random numbers coming from a normal distribution with mean 5 and standard deviation 2:

```
rnorm(6, 5, 2)
```

```
## [1]  3.943484  5.508093  5.707986 11.513517  6.817776  2.529775
```

And here is 5 random numbers coming from a uniform distribution between 0 and 1:

```
runif(5)
```

```
## [1] 0.3346454 0.4547205 0.2853403 0.5616378 0.3075359
```

10.6 Reproducing samples

Note that if you run any of these commands again you'll get a different sample. Sometimes you want your random sample to be reproducible i.e. you want to get the same random sample each time you run the script.
This is possible because the generation of randomness of any programming language is based on some algorithm that takes some "seed" number as input. It is usually based on the internal clock of your computer and so it changes all the time. But we can fix the seed manually by running the function `set.seed()` with an integer inside the parenthesis

QUESTION 3.1: Produce 5 random integers between 1 and 20, without replacement. Set a seed before sampling and confirm that you get the same sample each time you run it.

```
???
sample(???, ???)
```

10.7 Sampling from data

We can sample from data in the same way as above. Here I am sampling 10 random names from the ukbabynames dataframe:

```
sample(ukbabynames$name, 10)
```

```
##  [1] "Amisha"  "Abubakr" "Craig"   "Eirian"  "Aashir"  "Fisher"  "Melika"
##  [8] "Reza"    "Aleasha" "Phil"
```

Note also that the package dplyr contains its own functon sample_n() that allow you to sample the number of rows that you indicate as its second argument (which is the first argument if you use "%>%"):

```
library(dplyr)
ukbabynames %>% sample_n(8)
```

```
##   year sex     name  n rank
## 1 1998   F   Briege  3 3848
## 2 1998   M    Caner  8 1485
## 3 2004   M  Fardeen 13 1347
## 4 2006   F   Kassia  9 2320
## 5 2001   M   Tyreke  7 1746
## 6 2011   F    Kiswa  3 5785
## 7 2005   M    Nasif  5 2754
## 8 2000   F Annalisa  8 1995
```

10.8 Making functions

10.9 When to write your own functions

If there's already a function in R that does what you want then writing it yourself will almost definitely be less efficient as well as wasting your time. For example, writing a function to calculate the standard deviation of a column is not needed since there is already the `sd()` function.

Many common things will already have function either in base R or a package, however if you find yourself doing the same specific task multiple times, writing your own function can save you copying and pasting code and this is always desirable.

We define a function that way:

```
myFunction <- function(){

}
```

and use it like any other function

```
myFunction()
```

```
## NULL
```

Yes for now our function doesn't do anything, because there is nothing inside the brakets (note tha they are curly brackets by the way, not normal or square brackets!).

We can make the function return something, with the function return()

```r
myFunction <- function(){
  return(2)
}
myFunction()
```

```
## [1] 2
```

Our function just returns two each time we call it. What an awesome function!!!

Let's put inside some code that generates the sum of 3 uniformely distributed random numbers and let's return that sum

```r
myFunction <- function(){
  few_runif <- runif(3)
  sum_few_runif <- sum(few_runif)
  return(sum_few_runif)
}
myFunction()
```

```
## [1] 2.253677
```

Yayy! Did you see that each time you run it you get a different number?

QUESTION 4.1: Create a function called myFunctionNorm that would return the sum of 5 normally distributed random numbers. Run it to see if it works

```r
??? <- function(){
  few_rnorm <- ???(???)
  sum_few_rnorm <- ???(???)
  return(???)
}

???()
```

You may be tempted to put a number inside the parenthesis. Would myFunction(5) generate 5 random numbers?

```r
myFunction(5)
```

Argh, no, because we didn't define what the function should do with what is inside the parenthesis.

We can generate more than one number with the function replicate() though, which simply runs what we put as its second argument (second thing inside the parentheses), the number of times we indicate as its first argument:

```r
randomNumbers <- replicate(5, myFunction())
randomNumbers
```

```
## [1] 1.6814483 0.8994199 1.0328035 1.5335543 0.5780411
```

If we felt that the command replicate(number, myFunction()) was a bit too long to write, we could make a function similar to rnorm(), runif() etc. that gives the size of the sample as the first argument:

```r
rsum3unif <- function(n){
  some_sum3unif <- replicate(n,myFunction())
  return(some_sum3unif)
}

rsum3unif(8)
```

```
## [1] 1.6704208 1.1739316 0.9551181 1.5962373 1.8420413 1.7429733 1.5424678
## [8] 1.7029722
```

And here it is, we made a function that takes a number as its first (and only) argument, which it then used as the first argument of the function replicate, to generate the required number of random numbers (made from the sum of 3 uniformely distributed reandom numbers).

QUESTION 4.2: the function below is supposed to print in the console a smiley that says the words given as its argument. What would you need to write in place of ??? for it to work? (note: the function cat() prints what's inside its parenthesis on the screen. "\\n" tells R to go to the next line). Try it

```r
smileysay <- function(???){
  cat("----------\n")
  cat(words)
  cat("\n----------")
  cat("\n       ")
  cat("\n ^   ^ ")
  cat("\n   |   ")
  cat("\n  ___  ")
  cat("\n |   | ")
  cat("\n  ---  ")
}

smileysay("Hello R!")
```

10.10 Iterations (loops)

Here is a simple loop for:

```r
for(i in 3:9) {
  print(i)
}
```

```
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
```

The first part: (i in 3:9) provides the set of values to loop over. What is inside the curly brackets are the things that we are asking R to do at each iteration of the loop. Here we are asking to print i.

So we can read the above code as "i will iteratively take all the values from 3 to 9 and for each iteration i should be printed on the screen".

Instead of 3:9 we could put any kind of list of elements, and we don't need i to be called i. For example let's loop over a list of 10 random numbers, with the iteration variable called rnumber instead of i:

```r
randomNumbers<- runif(10)
for(rnumber in randomNumbers) {
  print(rnumber)
}
```

```
## [1] 0.150905
## [1] 0.07981697
## [1] 0.7518809
## [1] 0.6252572
## [1] 0.3606134
## [1] 0.6199747
## [1] 0.9494314
## [1] 0.5214505
## [1] 0.715805
## [1] 0.5418952
```

QUESTION 5.1: Make a loop with 12 iterations that generates and print at each iteration a random number coming from a standard normal distribution . Call the iteration variable "iter".

```
for(??? in 1:???) {
  random_number <- ???(1)
  print(???)
}
```

In addition to being able to make loops, we will need to know how to check whether the name found in a box is or not the the name of the prisoner. We will do that with the function if()

If() works like the function for(), but is a bit simpler: if what is inside the parenthesis is true, then what is inside the curly brackets is performed. Let's generate a random number and check if it is larger than 0:

```
aNummber <- rnorm(1)
aNummber
```

```
## [1] -1.433584
```

```
if(aNummber>0) {
  print("it is larger than 0")
}
```

If you run the above code multiple times, you will see that sometimes something will be printed and sometimes not.

We can also ask R to perform an action when the condition inside the parenthesis is false, using the else command:

```
aNummber <- rnorm(1)
aNummber
```

```
## [1] 2.441561
```

```
if(aNummber>0) {
  print("it is larger than 0")
} else{
  print("it is negative!")
}
```

```
## [1] "it is larger than 0"
```

Also note that there exists another function called "ifelse()" that allows us to check a same condition on multiple elements at the same time. It works more like a normal function, the first argument being the condition to check, the second is what the function must return if the condition is checked and the third, what it must return if it is not. Let's generate 10 random numbers, check whether they are greater than 0, and for each of them, return 1 if this condtion is true and 0 if it is not:

```
fewNummbers <- rnorm(10)
fewNummbers
```

```
##  [1] -1.0278143 -0.5947616  1.7244637  0.8588215 -0.1388179  0.2179455
##  [7] -0.1130976  1.1966227  0.3479995  0.4829627
```

```
ifelse(fewNummbers>0, 1, 0)
```

```
##  [1] 0 0 1 1 0 1 0 1 1 1
```

Do you see that you could achieve the same thing by looping over the 10 random number? Actually, R contains lots of functions like ifelse() that we could replace with loops but these functions are optimize and therefore much faster than using loops. Very often you'll be able to avoid using loops, and you should try so. They will only be unavoidable in some very specific situations.

QUESTION 5.2: can you produce the same sequence of 0 and 1 as above, but using a loop for() and the if() function?

```
for(i in ???){
  if(???){
    print(???)
  }
  else{
    print(???)
  }
}
```

10.11 Puttng it all together

Here is a version of a very hard probability problem, with a very unintuitive result, that was first published by Peter Bro Miltersen and Anna Gal in the Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP):

The names of 100 prisoners are placed in 100 wooden boxes, one name to a box, and the boxes are lined up and numbered on a table in a room. One by one, the prisoners are led into the room; each may look in at most 50 boxes, but must leave the room exactly as he found it and is permitted no further communication with the others. The prisoners have a chance to plot their strategy in advance, and they are going to need it, because unless every single prisoner finds his own name all will subsequently be executed.

QUESTION Can you guess what is the probability of success of the best strategy?

One simple strategy would be for each prisoner to open 50 boxes at random. Since there are 100 boxes in total, each prisoner will have a probability of 1/2 to open the box that contains his name. Since each prisoner chooses randomly, the probability of success of the prisoners are independant and so the probability that all the prisoners will manage to open the box that contains their name is:

```
(1/2)^100
```

```
## [1] 7.888609e-31
```

… quite low!

Now the best strategy is quite simple, but understanding why it is the best is not at all obvious. It requires quite heavy math thinking about permutations and cycles. The same goes for calculating the associated probability of success. What we can do though is make some simulations to get an approximation of it.

The best strategy is the following: First, the prisoners should agree on a random assignment of each box to a prisoner's name. We can imagine they all have in mind an imaginary common labelling of the 100 boxes.

Then each prisoner opens the box assigned to him. If the box contains his name, great. Otherwise, he opens the box labelled with the name found in this first box, and then he opens the box labelled with the name found in the second box etc. until he has either found his name, or opened 50 boxes.

10.12 Simulation of the best strategy

First, we need 100 names for our prisoners though. Here they are:

```
set.seed(79)
list_prisoners <- sample(ukbabynames$name, 100)
list_prisoners
```

```
##   [1] "Abela"       "Taher"       "Fynlay"      "Gisele"      "Wania"
##   [6] "Adelaide"    "Mansoor"     "Larissa"     "Hayley"      "Iona"
##  [11] "Anesh"       "Ashir"       "Luisa"       "Shakti"      "Scarlett"
##  [16] "Evie"        "Tiger"       "Geoffrey"    "Cassie"      "Shariq"
##  [21] "Alarna"      "Rawdah"      "Hadassah"    "Ikram"       "Amia"
##  [26] "Jamieleigh"  "Demilade"    "Kovan"       "Demi-Rae"    "Azim"
##  [31] "Gaurav"      "Christos"    "Jarin"       "Domenico"    "Manaal"
##  [36] "Elijah"      "Katie-Mae"   "Abdulhakeem" "Krystal"     "Nana"
##  [41] "Stacie"      "Yaw"         "Kay"         "Lily-Ann"    "Sola"
##  [46] "Iara"        "Honie"       "Liana"       "George"      "Kaylem"
##  [51] "Khushal"     "Bushra"      "Yasa"        "Filipa"      "Braidon"
##  [56] "Lois"        "Daisy-may"   "Esmay"       "Alisdair"    "Sunaina"
##  [61] "Muhanad"     "Huw"         "Taegan"      "Catelynn"    "Byron"
##  [66] "Gage"        "Delphi"      "Faiz"        "Shaam"       "Zafir"
##  [71] "Rivaan"      "Haajira"     "Amye"        "Warrick"     "Sajan"
##  [76] "Ephraim"     "Evie-Lea"    "Ayush"       "Maisie"      "Ugochi"
##  [81] "Samad"       "Bhavika"     "Delara"      "Monet"       "Kalani"
##  [86] "Aashi"       "Hajira"      "Ishika"      "Isra"        "Kerensa"
##  [91] "Samy"        "Mariam"      "Tinotenda"   "Shiloh"      "Glory"
##  [96] "Diego"       "Dayton"      "Jeorgie"     "Promise"     "Payal"
```

What we will need to do then is sample from this list to randomly place the names of the prisoners in the boxes:

```r
name_inside_box <- sample(list_prisoners)
name_inside_box[1:10]
```

```
##  [1] "Gaurav"   "Jeorgie"  "Ashir"    "Bhavika"  "Dayton"   "Azim"
##  [7] "Jarin"    "Amia"     "Hadassah" "Stacie"
```

So for this simulation, in the first box there is the name `name_inside_box[1]`, in the second one, `name_inside_box[2]` etc.

And we can do the same to simulate the imaginary random labelling of the boxes by the prisoners when they develop their strategy:

```r
label_on_box <- sample(list_prisoners)
label_on_box[1:10]
```

```
##  [1] "Haajira"  "Ishika"   "Lily-Ann" "Delara"   "Ashir"    "Kerensa"
##  [7] "Demilade" "Tiger"    "Bhavika"  "Evie"
```

And so `label_on_box[1]` will open the first box, where he will find the name `name_inside_box[1]`, which is not his own name, so he will look for the box labelled `name_inside_box[1]`.

We can find this box with the function `which()`:

```
box_to_open <- which(label_on_box==name_inside_box[1])
box_to_open
```

```
## [1] 49
```

So he will look in the box `box_to_open` where there is the name

```
name_inside_box[box_to_open]
```

```
## [1] "Kay"
```

Etc. etc.

We definitely need to make a loop here… and we also need a function that will encapsulate all our code, so that that we can easily do many simulations using replicate().

Here is our empty function. Note that from now on, most of the code below won't return anything, but try to follow how the function is filled.

```
simu_game <- function(){

}
```

We start by initiating the game and strategy by making a random assignment of names in the boxes and a random assignment of imaginary labels on the boxes:

```
simu_game <- function(){
  name_inside_box <- sample(list_prisoners)
  label_on_box <- sample(list_prisoners)
}
```

We will then need two `for()` loops, one to loop over the prisoners and one to loop over the attempts

```
simu_game <- function(){
  name_inside_box <- sample(list_prisoners)
  label_on_box <- sample(list_prisoners)
  for(prisoner in list_prisoners){
    for(attempt in 1:50){

    }
  }
}
```

At each attempt, we will look at the name that is inside the box that the prisoner needs to open, and check whether it is or not the name of the prisoner:

```
simu_game <- function(){
  name_inside_box <- sample(list_prisoners)
  label_on_box <- sample(list_prisoners)
  for(prisoner in list_prisoners){
    for(attempt in 1:50){
      name_found_in_box <- name_inside_box[box_to_open]
      if(name_found_in_box==prisoner){

      }
    }
  }
}
```

For now we didn't define what is the box to be opened. At the beginning it is the box labelled with the name of the prisoner. After the first attempt, it is the box labelled with the name found in the previous box

```
simu_game <- function(){
  name_inside_box <- sample(list_prisoners)
  label_on_box <- sample(list_prisoners)
  for(prisoner in list_prisoners){
    box_to_open <- which(label_on_box==prisoner)
    for(attempt in 1:50){
      name_found_in_box <- name_inside_box[box_to_open]
      if(name_found_in_box==prisoner){

      }
      box_to_open <- which(label_on_box==name_found_in_box)
    }
  }
}
```

Let's now add a variable called "name_found". Let's make it equal to FALSE for each prisoner at the beginning and let's change it to TRUE if the name found in a box is the name of the prisoner:

```
simu_game <- function(){
  name_inside_box <- sample(list_prisoners)
  label_on_box <- sample(list_prisoners)
  for(prisoner in list_prisoners){
    name_found <- FALSE
    box_to_open <- which(label_on_box==prisoner)
    for(attempt in 1:50){
      name_found_in_box <- name_inside_box[box_to_open]
      if(name_found_in_box==prisoner){
        name_found <-TRUE
      }
      box_to_open <- which(label_on_box==name_found_in_box)
    }
  }
}
```

And let's add one last variable called "success". It will be equal to 1 at the very beginning and changed to 0 if a prisoner doesn't find their name:

```
simu_game <- function(){
  name_inside_box <- sample(list_prisoners)
  label_on_box <- sample(list_prisoners)
  success <- 1
  for(prisoner in list_prisoners){
    name_found <- FALSE
    box_to_open <- which(label_on_box==prisoner)
    for(attempt in 1:50){
      name_found_in_box <- name_inside_box[box_to_open]
      if(name_found_in_box==prisoner){
        name_found <-TRUE
      }
      box_to_open <- which(label_on_box==name_found_in_box)
    }
    if(name_found==FALSE){
      success <- 0
    }
  }
}
```

Ok I think we're done now. At the very end, we just need to return the value of "success".

```
simu_game <- function(){
  name_inside_box <- sample(list_prisoners)
  label_on_box <- sample(list_prisoners)
  success <- 1
  for(prisoner in list_prisoners){
    name_found <- FALSE
    box_to_open <- which(label_on_box==prisoner)
    for(attempt in 1:50){
      name_found_in_box <- name_inside_box[box_to_open]
      if(name_found_in_box==prisoner){
        name_found <-TRUE
      }
      box_to_open <- which(label_on_box==name_found_in_box)
    }
    if(name_found==FALSE){
      success <- 0
    }
  }
  return(success)
}
```

Let's try our function:

```
simu_game()
```

```
## [1] 0
```

And now let's replicate it a hundred times and make the sum of all the success

```
simulations <- replicate(100,simu_game())

sum(simulations)
```

```
## [1] 30
```

Out of our 100 simulations, the strategy was successful 30 times! If we compare it to the probability of success of the completely random strategy, that is a huge improvement!

QUESTION: Could you modify our simu_game function so that it takes the number of boxes that each prisoner can open as its argument. How often does the strategy seem to work if each prisoner can open 75 boxes (there are three ??? that you need to replace)

```
simu_game <- function(???){
  name_inside_box <- sample(list_prisoners)
  label_on_box <- sample(list_prisoners)
  success <- 1
  for(prisoner in list_prisoners){
    name_found <- FALSE
    box_to_open <- which(label_on_box==prisoner)
    for(attempt in 1:???){
      name_found_in_box <- name_inside_box[box_to_open]
      if(name_found_in_box==prisoner){
        name_found <-TRUE
      }
      box_to_open <- which(label_on_box==name_found_in_box)
    }
    if(name_found==FALSE){
      success <- 0
    }
  }
  return(success)
}

simulations <- replicate(100,simu_game(???))
sum(simulations)
```

BONUS QUESTION: We could be a bit more efficient by using while() loops instead of for() loops. Can you find how to modify the code to use while() loops? (Note: You just need conditions inside the parenthesis of while(). You do need to add a few lines though to make it work. these lines are added with ??? below)

```
simu_game <- function(){
  name_inside_box <- sample(list_prisoners)
  label_on_box <- sample(list_prisoners)
  success <- 1
  ???<-1
  while(??? & ???){
    prisoner <- list_prisoners[i]
    name_found <- FALSE
    box_to_open <- which(label_on_box==prisoner)
    ???<-1
    while(??? & ???){
      name_found_in_box <- name_inside_box[box_to_open]
      if(name_found_in_box==prisoner){
        name_found <-TRUE
      }
      box_to_open <- which(label_on_box==name_found_in_box)
      ??? <-???+1
    }
    if(name_found==FALSE){
      success <- 0
    }
    ??? <- ???+1
  }
  return(success)
}

simulations <- replicate(100,simu_game())
sum(simulations)
```

Bibliography

Cards Against Humanity (2018). Pulse of the Nation. https://thepulseofthenation.com/downloads/201806-CAH_PulseOfTheNation_Raw.csv.

FAOSTAT (2018). Live Stock. http://www.fao.org/faostat/en/#data/QA.

IMDB (2018). Imdb. https://www.imdb.com/interfaces/.

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., and Woo, K. (2019a). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.1.1.

Wickham, H., François, R., Henry, L., and Müller, K. (2019b). *dplyr: A Grammar of Data Manipulation*. R package version 0.8.1.