Fundamentals of Data Science

Paul King

19 September 2023

Table of contents

Getting Set Up

Welcome to Fundamentals of Data Science (MA7419 / MA3419)

Congratulations. By the end of this module you will be confident in handling data in a collaborative and reproducible way. In other words in a **professional** way.

This is a vital skill for almost any of the jobs you're likely to move on to, actuarial or not. The module is for anyone who wants to be a professional (in all the good senses of that word) data scientist.

There's not going to be a lot of listening to lectures. Instead you will be developing the hands-on R programming skills to solve real data problems. And to do that, you'll need the right software - so that's the first step. We'll be using R (R Core Team 2022) and RStudio (Posit team 2023), two amazing - and free - programs.

Installing R and RStudio

If you have a suitable laptop, you're probably going to want to install R and RStudio on your own machine. This is good experience and lets you play around absolutely as you want to. Instructions for that are given below and I also strongly suggest you set up the project and folder structure described in the study material for Week 1.

You can also run RStudio on University PCs.

For most of the work in this module we will be working on a University of Leicester server which you can access from any device that can run a browser. So if you use a Chromebook you will be able to work just as well as people with machines running Windows or Mac OS. You'll also be able to do a certain amount on a decent sized tablet and you'll have access via a phone - though I think it would be hard to do any significant work that way.

Once your account has been set up (at the beginning of the semester) you should be able to access the server here:

https://rserver.mcs.le.ac.uk/rstudio/

You can then log in using your usual University user name (e.g. pk255) and your University password.

There is a final option. You could investigate Posit Cloud. Creating a free account there should give you everything you need, again through a browser. The disadvantage is that you only get 15 hours (last time I looked) per month for free. However, it can be useful as

a last resort if the other options fail (especially if you've kept good backups of your work so you can upload them and carry on seamlessly). If you decide to make an account don't use your university password on this, or any other cloud accounts.

We'll make sure everyone is up and running in the first week's sessions, so don't panic if things don't work immediately.

Please bring your laptop to lectures, that way you'll be able to follow along with what I'm doing at the front of the room - however, we won't be able to provide individual support in lectures, that's what the computer lab sessions are for.

Installing R on your own machine.

The internet is overflowing with great, free R resources. We're going to be using a number of them as the main reading for the module, and for instructions on installing software we might as well dive straight into an on-line book we'll use a lot: R for Data Science (Wickham and Grolemund 2017).

Read Chapter 1, *Introduction* (confusingly, Chapter 2 is also called *Introduction!*). And follow the instructions you'll find there to install R, RStudio, and the Tidyverse family of packages.

You might also want to start reading R Programming for Data Science (Roger D. Peng 2020). There are instructions for installing the software in Chapter 3 (please ignore Section 3.2).

DataCamp

DataCamp is a website containing a very large number of programming courses - including many on R.

Subscriptions are usually about 30 USD per month but we have arranged for you to have free access for the duration of this module.

Using the link you can find in the Blackboard site you can register for an account and use the resources there to improve your R skills. (You will need your university email address for this one, but do not use your University password.)

We've found that the DataCamp system can take a while to recognise you've signed in with a free account. You might well find that after completing a few free chapters you are invited to sign up for a paid subscription. **DO NOT DO THIS.** Instead you should just log out and go back in 24 hours, when you should find you can access everything for free.

Getting help

There are a lot of students on this module and just one lecturer and a few TAs. We want you to ask lots of questions, but if you email them all to us we will probably be overwhelmed so please post your questions on the Blackboard forum.

Code of Conduct

1. Help each other.
2. Be nice.
3. Acknowledge help you get from others. (If someone has been particularly helpful - or nice - let us all know on the forum or email me.)
4. Don't plagiarise, don't collaborate if you have been asked not to, and don't perform any other form of academic misconduct. You will almost certainly cut and paste code written by other people - acknowledge its source. The one exception is that you don't have to acknowledge the source of every bit of code you are given in this course - that could get tedious.
5. Respect other people's right to an undisturbed lecture.
6. Help each other.

Question for you: Is this a good list? What should be added or taken away?

1 Getting started with R

1.1 Overview

1.1.1 Objectives

On successful completion of this module, you should be able to:

- Define data science and discuss its role in actuarial science and business analytics
- Create and run simple program scripts in the RStudio environment and read data into R from a local tabular file
- Create and share literate programs and reports using R Markdown and undertake simple EDA (including summary statistics and informative visualisations) of data in a tabular structure
- Implement a reproducible workflow for simple data science project and describe ethical and regulatory issues to be considered when undertaking a data science project

I prefer an alternative statement of the objectives:

By the end of this module you will be confident in handling data in a collaborative and reproducible way. In other words in a **professional** way. You will be able to:

- get
- clean
- check
- restructure
- filter, sort, join & summarise
- explore
- visualise…

…data

The specific objectives for this week are:

1. Be able to access your R Server account
2. Successfully upload Task 1 from Blackboard to the server
3. Complete the exercises in the Week 1 task
4. Knit to HTML and download HTML and RMD files
5. Submit RMD and HTML files for feedback on Blackboard

1.1.2 Problem-based learning

The structure of this module may be a bit different to what you have come across before.

In a "traditional" university lecture you arrive without having done any preparation, the lecturer gives you some information, and you go away to try and absorb it and probably do some exercises or homework.

In a "flipped" classroom you are supposed to have done reading, or used other resources, so that you have some familiarity with the material before the class. In class you focus on applying what you have learnt to some exercises.

This module is quite similar to a flipped classroom, but here, working on the task and learning what you need to successfully complete it will happen much more in parallel. I will give you a quick overview of some of the things you need to know each week, and there will be some indicative reading. But after that it is up to you (usually working as part of a team) to work out a good solution to the task.

This is much more like the way things happen in real life.

1.1.3 Introductions

- Dr Paul King FIA (Lecturer)
  - paul.king@leicester.ac.uk
  - No set office hours: email for an appointment
- Your friendly TAs
  - Ahmed Dogar - ajd67.le.ac.uk
  - Ibrahim Issahaku - iai5@le.ac.uk
  - Daniel Lozano Rojas - dlr10@le.ac.uk

1.1.4 Class times

- Tuesday 13:00 - 14:00, Attenborough Building Lecture Theatre 1
- Wednesday 12:00 - 13:00, On-line - see the Teams invite on Blackboard
- Thursday 13:00 - 14:00, Various computer labs

1.1.5 How this module works

- A certain amount of lecturing (mostly on Tuesdays)
- A lot of group problem solving (mostly on Wednesdays and Thursdays)
- A lot of self study: using the reading list and other on-line resources
- Additional resources:
  - Data Science Club (if you want to run one!)

  - DataCamp

- The weekly tasks for an important part of the learning activity on this module and the completed tasks will be a key part of the set of notes you develop

1.1.6 Group working

From next week you will be working in groups on a weekly coding task. The output will be a R Markdown notebook containing your code and results - each member of the group must produce their own notebook and be able to edit it and run it on the server or their own machine. Completed tasks (i.e. the PDF file) should be submitted by midnight on Sunday (UK time) in the week of the class.

1.1.7 Getting help on the weekly task

You can ask for help from your TA during the Wednesday computer lab. Outside of the lab (or even inside) you can post questions on the Blackboard discussion forum. Try to formulate your question as specifically as possible and give a code sample. That way the question can be answered in the thread and benefit everybody (other students are encouraged to answer questions). If you need more detailed help a TA will contact you via Teams.

I'll create a separate chat for general questions about the course. Please try to put as many of your questions here as possible so everyone can benefit, but if it's something a bit more personal, please feel free to email me (paul.king@leicester.ac.uk) directly. I'm happy to arrange a one to one meeting over Teams or face to face.

1.1.8 Submitting the weekly task

When you've completed the weekly task, please submit both your RMD and HTML files via the assignment links on Blackboard (to be set up).

Each exercise in the task will be graded on the following scale:

0. Nothing (significant) submitted
1. Exercise attempted but not working / incomplete
2. Nearly there
3. Full correct answer (note, there is no single correct answer)

These are designed to be a quick and easy way of giving feedback and allowing me to check that everyone is progressing OK. These "marks" will not count towards your final assessment for the module.

1.1.9 DataCamp

You have been sent an invitation email and there is more information in Getting Set Up.

It's a very good idea to take advantage of this resource while it's available.

1.1.10 Data Club

- Chance to practice and develop Data Science Skills in an informal atmosphere
- First meeting is Thursday 13 October 2022 at 17:00 (Teams meeting on Blackboard)
- The idea is to meetup and work on a data task collaboratively to develop skills
- I'll host the initial meeting, but after that it will need to be run by you, so volunteers are needed.
- More info at Thursday's meeting

1.1.11 How this module is assessed

- Coursework mid-semester (30%)
  - similar to problems solved in class; but
  - you must work on your own
  - available at the end of week 15 and due in two weeks later
- Final report and presentation (70%)
  - you will work on a group data science project and produce both a detailed report of the analysis and a presentation
  - the presentation must include slides (generated from R markdown) but they will be marked on the submitted document - you will not be required to give a stand up presentation
  - each individual will have to identify an aspect they worked on, for questions
  - some of the marks will be for demonstrating knowledge of the topics in the lectures after revision week
  - assessments for MA3419 and MA7419 are different
- Weekly Blackboard quiz
  - Each week there will be a Blackboard quiz asking if you feel you have achieved the specified objectives for that week. This is a chance to reflect on your work over the week and thereby reinforce your learning
- more details will be given when the first coursework is made available

1.2 Reading

R for Data Science (Wickham and Grolemund 2017):

- Chapter 4 *Workflow: basics*

R Programming for Data Science (Roger D. Peng 2020):

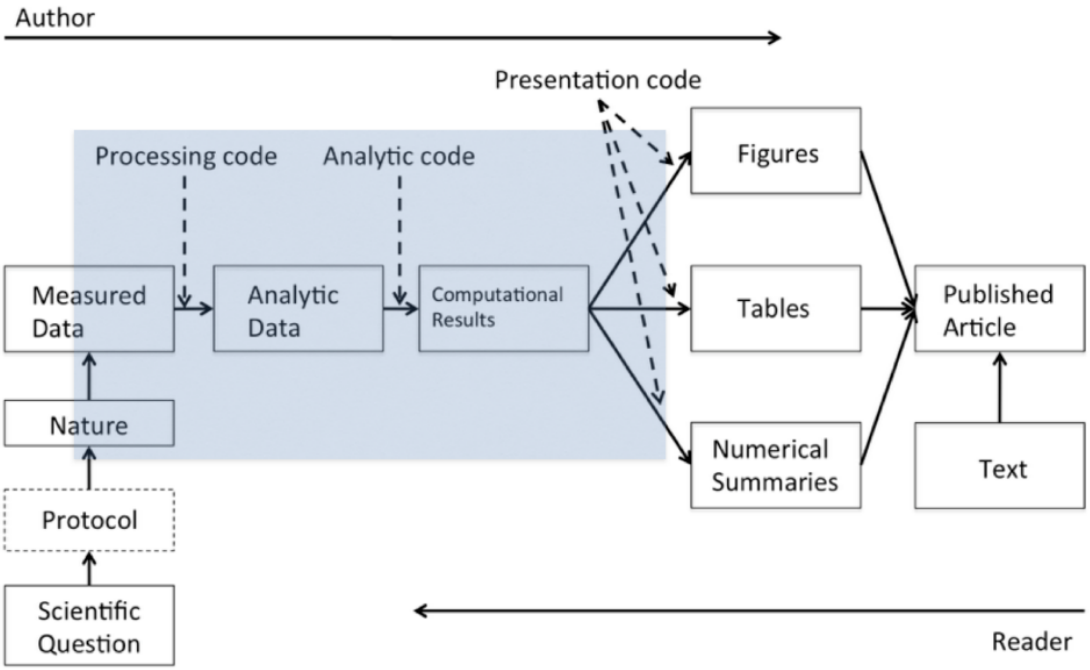
- Chapter 13 *R nuts & bolts*

1.3 Reproducible data science

1.3.1 Introduction

We are going to discuss *Reproducible Data Science*. This is often referred to as *Reproducible Research*, but it applies to lots of data science activities we might not consider as research.

It's helpful to consider the process of data science, for example as illustrated by Roger Peng in his book *Report Writing for Data Science in R* (Roger D. Peng 2019).



The steps in the shaded box start with the raw (measured) data and end with the computational results. In this module we'll be concerned with these steps - the core of a reproducible pipeline - plus the presentation code used to produce the final output.

In this diagram *analytic data* refers to the raw data after it has been checked, cleaned and reshaped into the right format for further analysis.

1.3.2 What is Reproducible Data Science?

A reproducible data science process allows a reviewer or subsequent user of the work to reproduce the analysis **at least** from the analytic data to the computational results. It should also allow them to:

- see what has been done at each stage;
- understand the reasons for the decisions made at each step;
- check that the claimed steps have been carried out correctly.

In all but the simplest of projects, this requires the publication of well written, and well documented, code.

When we consider published work in the context of a scientific publication, reproducibility, as a minimum, requires making available the analytic data, code and details of the computational environment used.

However, it's important the the whole process, including pre-processing the data and producing the final outputs, is fully documented.

1.3.3 Benefits (and disadvantages) of Reproducible Data Science

Scientific:

- Replication of published work is a key element of the scientific process: reproduction of the analysis of data analysis is often the closest to replication it's realistically possible;
- attempts at reproduction can reveal errors with potentially serious consequences.

Professional:

- Ability to reproduce a workflow is key to checking for errors;
- Documentation of data analysis is required for work review and auditability;
- For actuaries, the professional standards APS X2 and APS QA2 require consideration to be given to reproducibility in relation to the above points;
- A reproducible work flow allows efficient reuse or modification of an analysis pipeline.

Personal:

- The next user of your work is most likely to be you!

- the process of making your work reproducible requires you to stop (or at least slow down) occasionally to make sure you are working effectively and efficiently.

1.3.4 How to make sure your work is reproducible

- Create a consistent filing structure for data, code, intermediate and final results;
  - see the next section
- Use RStudio and `knitr` to practice *literate programming* by creating R Markdown notebooks;
- Resist the temptation to do things manually - do as much programmatically as possible
  - coding all your data manipulation is not just about automating it, it's about documenting what you've done so it can be reproduced;
- Comment code appropriately;
- Stick to a coding standard (style guide)
  - we will use the tidyverse style guide
- Save as few intermediate results as possible;
- Use version control;
- Keep track of your programming environment

The last two points are important, but we don't have the time to go into them in any detail in this course.

1.3.5 Further Reading

APS X2 Review of Actuarial Work

APS QA1 Quality Assurance for Organisations (Version 2.0)

The Institute and Faculty of Actuaries has written about how important Data Science is to the profession.

1.4 More resources

The Bookdown home page gives you access to many very good books covering different aspects of data science with R.

1.5 Setting up your R environment.

You will be working in R every week during this module, so the first task is to make sure you have access to RStudio.

The next step is to configure your environment in a standard way so it will be relatively straightforward for us all to work share code and work together.

Go ahead and:

1. Use RStudio to create a project called MA3419 or MA7419
2. Inside that project, create folders for each Week during the Semester, plus a folder called *Data*
    - here is a video demonstrating how to do steps 1 & 2.
3. Install the packages 'tidyverse' and 'here'
    - video demonstration
4. Get familiar with the RStudio interface
    - video
5. Get familiar with the R Markdown notebook structure and how to run code inside a notebook
    - video
    - I highly recommend this blog article
    - if you need more help, you'll almost certainly find something at RStudio
    - once you're reasonably familiar with RStudio and R Markdown have a look at some more advanced Tips and Tricks (This won't make much sense if you're an absolute beginner, but you should bookmark it and come back to it later.)

1.6 Week 1 Task

Every week there will be a task provided as a .RMD file in the Blackboard folder. You need to copy this file into the appropriate week's folder in your MA3419/MA7419 folder and have it open in a running R session during the Wednesday computer lab. The task files will become available on Monday mornings.

> **Note**
>
> The marks for the weekly task are for feedback only. They don't count towards your module assessment.

Sometimes there will also be a data file supplied - you should copy this into your project's Data folder.

Sometimes I'll work through parts of the task during the Tuesday lecture, and every week I'll give feedback on the most common mistakes in the previous week's task. There will also be exercises embedded in the notes for you to complete.

Your target should be to complete all the exercises in each weekly task, and submit your answers on Blackboard for feedback. I'll also release the correct code for the week's task at the end of the week.

Occasionally there will be *challenge exercises* or *challenge tasks* - these are optional.

2 Tabular data

2.1 Overview

This week we'll be learning about the data structures: vectors, data frames & tibbles.

Then we'll be using the `dplyr` package (part of `tidyverse`) to begin manipulating tabular data.

We'll be using the `dplyr` verbs select, filter, mutate and arrange.

By the end of this week you'll be able to:

1. Explore a tabular data set interactively in the console
2. Manipulate tabular data using the functions in `dplyr`
3. Read an external CSV or Excel file into R
4. Find the documentation for a function or package
5. Produce a simple scatter or bar chart in `ggplot2`
6. Draw a simple map with `ggplot` and add points
7. Output your work as a PDF file from RStudio

Believe it or not this small set of skills will enable you to do some very interesting exploratory data analysis (EDA)

2.2 Reading

R for Data Science (Wickham and Grolemund 2017): - Chapter 5 - *Data transformation*

R Programming for Data Science (Roger D. Peng 2020): - Chapter 5 - *Getting Data in and out of R* - Chapter 6 - *Using the readr package*

2.3 Digging deeper into the structure of a data frame

We're going to use the `starwars` dataset that is automatically loaded with the package `dplyr`. To get it you will have to make sure `dplyr` is loaded (It's one of the core `tidyverse` packages).

The first step in most analyses is to explore the data interactively in the console. The most common functions to do this are:

`View(df)`

`head(df)`

`summary(df)`

`glimpse(df)`

`str(df)`

You can watch a walk-through of the use of these functions on the starwars data.

Each all the elements of each column of a data frame must be of the same type. In `starwars` there are elements of type `numeric`, `character` and list.

Here is a demonstration of how to extract individual columns from a data frame.

2.4 Reading data from a tabular file

The most common way we'll be using to get data into R will be to load it from file - usually a CSV or Excel file.

To read a CSV file we will use the `read_csv` function which is part of the `readr` package loaded with `tidyverse` (There is a function `read.csv` which is part of base R, but `readr::read_csv` is better.)

There are lots of optional parameters that you can use to refine the performance of `read_csv`, but it often works fine with just the path to the file.

For example, if we have a file of the 100 most popular girl babies names in England and Wales, in a file called `GirlsNames.csv` we can import it with the following code.

```
library(dplyr)
library(readr)
library(here)
girls <- read_csv(here("_Data", "GirlsNames.csv"))
head(girls, 10)
```

```
# A tibble: 10 x 3
    Rank Name     Count
   <dbl> <chr>    <dbl>
 1     1 OLIVIA    3866
 2     2 AMELIA    3546
 3     3 ISLA      2830
 4     4 AVA       2805
 5     5 MIA       2368
 6     6 ISABELLA  2297
 7     7 GRACE     2242
```

```
 8     8 SOPHIA    2236
 9     9 LILY      2181
10    10 EMILY     2150
```

(Data downloaded from the ONS)

Questions for you: Is this the most up to date list? Can you find a similar list from another country?

Data in Excel files can be read in similarly.

```
library(readxl)
cities <- read_excel(here("_Data", "UK_cities.xlsx"))
cities %>%
  arrange(Latitude) %>%
  head(10)
```

```
# A tibble: 10 x 4
   City            County      Latitude Longitude
   <chr>           <chr>          <dbl>     <dbl>
 1 Truro           Cornwall        50.3    -5.05
 2 Plymouth        UK              50.4    -4.14
 3 Exeter          the UK          50.7    -3.53
 4 Bournemouth     UK              50.7    -1.90
 5 Eastbourne      East Sussex     50.8     0.290
 6 Portsmouth      Hampshire       50.8    -1.09
 7 Worthing        West Sussex     50.8    -0.384
 8 Brighton & Hove East Sussex     50.8    -0.153
 9 Chichester      West Sussex     50.8    -0.779
10 Hastings        East Sussex     50.9     0.573
```

2.5 Documentation and help

You should get into the habit of looking at the documentation for each function the first time you use it.

The first place to look is by using the built-in help in RStudio. You can either go to the help window and use the search, or you can type "?" in the console. For example "?read_csv". (You'll usually need to have the relevant package loaded to get help, but "??" might produce something useful.). The help information can sometimes be a bit technical and overwhelming, but there are usually helpful examples at the end.

Have a look at the help page for `read_csv` now.

The official repository for packages is called CRAN. There you will find the package documentation which always has a PDF reference document with details of all the functions in

the package. You'll also often find one or more *vignettes* which are tutorial-style documents giving an introduction to the package and maybe more detail on particular aspects.

Try a web search for "CRAN dplyr" now and see what you can find.

Of course, there is a lot of other support available online. You can try the tidyverse site or stack overflow

2.6 Simple plots with ggplot

Now we know how to get data into R it won't be long before you want to plot it.

There are a number of alternatives but we'll be using `ggplot`. The syntax can take a bit of getting used to, so here are a couple of simple examples.

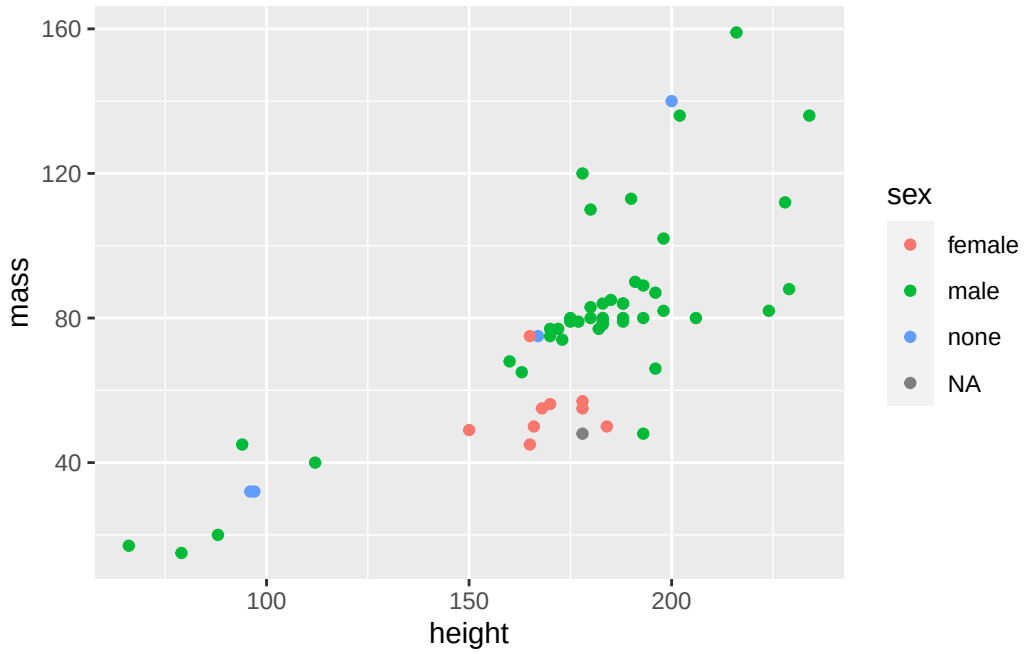
There are three different components to making a plot in `ggplot`.

1. a data frame containing the data you want to plot
2. the type of plot you want
3. the columns containing the data to be plotted

Here's the simplest possible example using the `starwars` data. (here's a walk-through)

```
library(ggplot2)
plot_data <-
  starwars %>%
  filter(mass < 1000) # this is explained in the walk-through

ggplot(plot_data) +
  geom_point(aes(x = height, y = mass, color = sex))
```



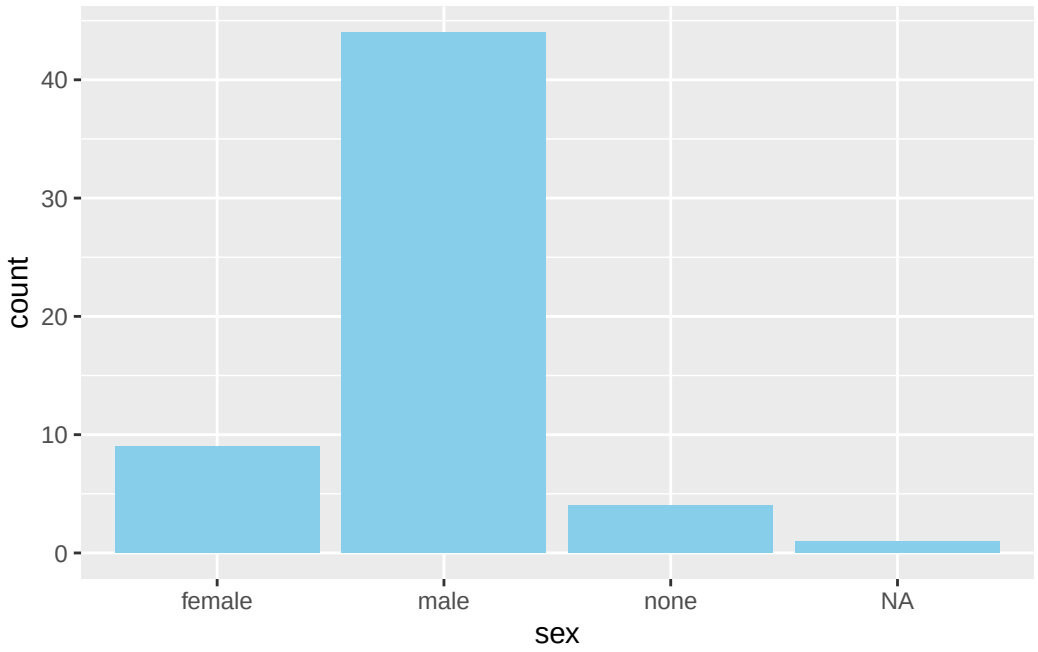
Part 1 is dealt with by passing the data frame as a parameter to `ggplot`.

Part 2 is dealt with by choosing an appropriate `geom_` function.

Part 3 is dealt with by the parameters to the `aes` function. Note that different geoms have different `aes` requirements. See the built in help for the particular geom.

Here's another example:

```
ggplot(plot_data) +
  geom_bar(aes(x = sex), fill = "skyblue")
```



For a quick and easy guide to producing many types of plots see the R Graphics Cookbook (Chang 2020).

2.7 Simple maps with ggplot

There's no denying that manipulating and visualizing spatial data can be very complex and most of the techniques are far beyond the scope of this course. But it's such a powerful visualization method that I thought it was important to give you a way to get started by plotting points on a simple map.

If you want to find out more about this topic I recommend Geocomputation with R (Robin Lovelace 2020)

There are many formats for storing spatial data. In this example we use two: shapefiles and the relatively new Simple Features,

Shapefiles are well established, and most publishers of geographical information will make it available in this format (possibly alongside others)

The simple features format allows us to work with geographical data in the familiar form of a data frame, and to plot maps using `ggplot`. We'll need to install the `sf` package before we can get started.

We are going to use low resolution country boundary shapefiles from here. (I've already downloaded files for the UK and Ireland and put them in Blackboard.)

The first bit of code reads in two files and converts them to `sf` objects.

```
library(sf)
Ireland <-
  st_read(here("_Data", "Ireland_Boundaries-shp", "Country_Boundaries.shp"),
          quiet = TRUE)
UK <-
  st_read(here("_Data", "UK_Boundaries-shp", "Country_Boundaries.shp"),
          quiet = TRUE)
```

Let's look at what's inside one of these `sf` objects.

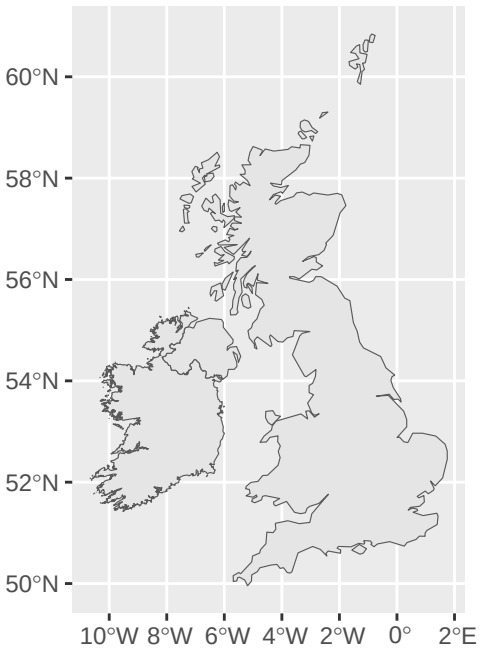
```
glimpse(Ireland)
```

```
Rows: 1
Columns: 8
$ OBJECTID_1 <int> 104
$ OBJECTID   <int> 104
$ name       <chr> "Ireland"
$ Id         <int> 0
$ Shape_Leng <dbl> 71.55104
$ Shape_Le_1 <dbl> 71.55104
$ Shape_Area <dbl> 9.443232
$ geometry   <MULTIPOLYGON [°]> MULTIPOLYGON (((-9.46639 51...
```

We can see there is only one row and 8 columns. The row contains attributes and geometry information about a single feature - crucially it contains geometry information which, in this case defines a set of polygons which make up the boundary of Ireland.

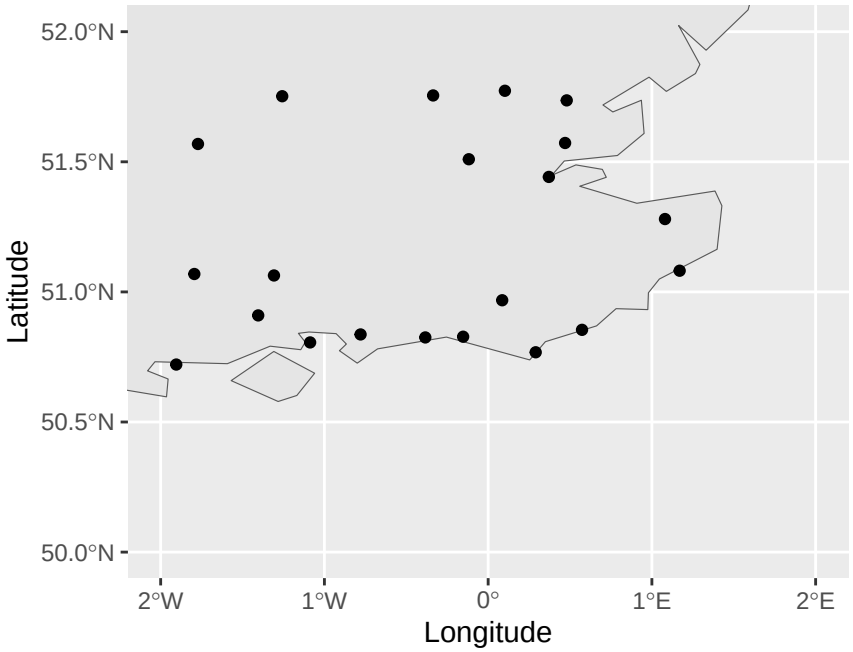
Since they came from the same source, the UK file contains the same columns, so we can stick them together to create a single object which we can then plot using the `sf_geom` provided by `ggplot2`

```
UK_IRL <- bind_rows(UK, Ireland)
m <- ggplot() + geom_sf(data = UK_IRL) # We can plot m now, and add to it later.
m
```



To demonstrate how we can add additional points to this data, we'll add the cities we listed above to the map. We'll also change the limits of the plot to zoom in on the South East of the country.

```
m + geom_point(data = cities, aes(x = Longitude, y = Latitude)) +
  xlim(c(-2, 2)) +
  ylim(c(50, 52))
```



At this scale we can see the low resolution of the boundary information.

2.8 Week 2 Task

The task for Week 2 can be found in the Week 2 folder on Blackboard.

3 Group and summarise

3.1 Overview

This week we'll be doing more manipulation of tabular data using the `dplyr`(Wickham et al. 2023) verbs `group()` and `summarise()`.

We'll also be creating more bar charts using `ggplot2`(Wickham 2016).

By the end of this week you'll be able to:

1. Group and summarise data in a tabular data set
2. Use `table()` and functions from the `janitor` package to create cross-tab summaries
3. Use functions from the `forcats` package to work with factors

You'll need to use the references and the R help system to develop the above skills as you work through the Week 3 task.

3.2 Reading

R for Data Science (Wickham and Grolemund 2017):

- Chapter 3 *Data visualisation*
- Chapter 5 *Data transformation*
- Chapter 6 *Workflow: scripts*
- Chapter 7 *Exploratory Data Analysis*
- Chapter 8 *Workflow: projects*

R Programming for Data Science (Roger D. Peng 2020):

- Chapter 12 *Managing Data Frames with the dplyr package*

3.3 Making slides in R Markdown

You'll need to make slides for the final assessment - so here's a preview. It's very easy - you can do it straight from the menu:

File > New File > R Markdown...

See the guide to producing slides (part of a larger introduction to using R Markdown from the makers of RStudio.)

3.4 Debugging

Code almost never works properly the first time you write it

Running into bugs and errors is frustrating, but it's also an opportunity to learn a bit more.

Errors can be obscure, but they are usually not malicious or random.

If something has gone wrong, you can find out why it happened.

3.5 Knitting to HTML (hints)

1. make sure all your packages are up to date.
2. does all your .RMD code run properly (i.e. if you click "run all" does it run without any error messages) - if it doesn't, fix it.
3. if (2) works, start a new R session on the server (for example, by closing down and restarting) - does the RMD code still run? If not, fix it.
4. once the RMD file runs, try clicking on knit to HTML, again, if that fails and you can't fix it, ask for help.
5. If you can knit to HTMl, you can try knitting to Word and PDF.
6. If you can knit to Word, you can create a PDF file by exporting from Word
7. If you can't create a PDF file in any other way you can try this as a last resort. If knitting to HTML works and opens an HTML file in a viewer window, click on "view in browser". If you are using Chrome, you will be able to save as a PDF document under "print". I advise installing Chrome for UoL work because that is the standard on University machines so most things are tested on it.

3.6 Week 3 Task

Some bigger files to work on this week, and some wider research required.

The task is on Blackboard and on the server at

`/data/FDS/Scripts/`

`/data/FDS/Data/`

Check your understanding

What is a "vignette" in the world of R?

A vignette is a readble introduction to a package, or a particular aspect of it. It is typically more friendly than the bare package documentation (such as the reference PDF file found on CRAN) and contains code examples in a tutorial style.

Find the vignette called *Introduction to dplyr* and study it.

Here's the link to Introduction to dplyr.

4 Working with tidy data

4.1 Overview

This week we'll be examining the concept of tidy data and looking at how to pivot between different data formats.

We'll also be looking at an important method of searching and manipulating strings called regular expressions.

By the end of this week you'll be able to:

1. Define the concept of tidy data and describe when it's useful
2. Convert between narrower (tidy) data frames and wider ones using the functions in `tidyr`
3. Describe the purpose and structure of a regular expression
4. Use regular expressions in conjunction with the `stringr` package

As a special bonus, you'll be able to use regular expressions for searching inside Microsoft Office documents.

4.2 Tidy data

Tidy data is data where:

- Every column is a variable.
- Every row is an observation..
- Every cell is a single value.

Tidy data describes a standard way of storing data that is used wherever possible throughout the tidyverse. If you ensure that your data is tidy, you'll spend less time fighting with the tools and more time working on your analysis.

However, tidy data isn't always the best solution. In particular humans usually find wider data frames easier to take in and understand, so for presentation and data entry tasks the wider format is often best.

Here's an example of a small tidy data set:

```
# A tibble: 6 x 3
  Name     Test  Score
  <chr>   <int>  <dbl>
1 Tiddles     1 -0.527
2 Rover       1 -1.76
3 Tiddles     2 -0.913
4 Rover       2 -0.683
5 Tiddles     3  0.474
6 Rover       3  0.697
```

And here's the same data in a non-tidy (wide) format.

```
wide_df <-
  tidy_df %>%
    pivot_wider(names_from = Test,
                names_prefix = "Test_",
                values_from = Score)

wide_df
```

```
# A tibble: 2 x 4
  Name    Test_1 Test_2 Test_3
  <chr>    <dbl>  <dbl>  <dbl>
1 Tiddles -0.527 -0.913  0.474
2 Rover   -1.76  -0.683  0.697
```

The functions in the `tidyverse` (notably `ggplot`) expect data in a tidy format but the wider format is often easier for people to read or to use when entering data.

As you've just seen, the `pivot_wider()` function takes us from tidy (aka "longer") format to a wider format. We can go back with `pivot_longer()`. Notice that the parameters are matched to those in the 'pivot wider()' example above.

```
wide_df %>%
  pivot_longer(cols = !Name,              # Apply to all columns except Name
               names_to = "Test",
               names_prefix = "Test_",    # Remove this from the names.
               values_to = "Score",
               )
```

```
# A tibble: 6 x 3
  Name    Test   Score
  <chr>   <chr>  <dbl>
1 Tiddles 1     -0.527
```

```
2 Tiddles 2     -0.913
3 Tiddles 3      0.474
4 Rover   1     -1.76
5 Rover   2     -0.683
6 Rover   3      0.697
```

For more details, see the pivot vignette.

Note: `pivot_wider()` and `pivot_longer()` replaced the previous functions `spread()` and `gather()`, which you may still see around.

4.3 Regular expressions

Regular expressions can be thought of as a mechanism for advanced search. They provide a language for writing a pattern which is used to match occurrences withing a character string. This ability is made use of by functions in many different languages - in R we'll mainly use them in conjunction with the 'stringr' package.

Here's a simple example. Suppose the string we want to search is "Paul wrote this in 2020."

The simplest search would be to see if `target` contains the string "Paul", like this.

```
library(stringr)
target <- "Paul wrote this in 2020."
target %>%
  str_detect(pattern = "Paul")
```

```
[1] TRUE
```

Or we could extract the year contained in the target using a pattern that finds any string of exactly 4 digits.

```
target %>%
  str_extract(pattern = "\\d{4}")
```

```
[1] "2020"
```

You'll see that a regular expression (or regex) will often contain a special character like `\\d`, which matches any digit. The double backslash is a quirk of R. Most implementations of regular expressions only require a single backslash, so if you use any resource not written specifically for R you'll see `\d` rather than `\\d`.

In fact (rather like SQL) there are minor differences in the implementation across different languages: something to bear in mind if you get stuck debugging some regex code.

Our treatment in the lecture is by no means comprehensive - regular expressions can be considered a language in their own right. See:

Regular expressions

My favourite quick reference is:

Regex Cheat Sheet

For `stringr` see:

Introduction to stringr

The university library has a book called *Mastering Regular Expressions* (Friedl 2006).

4.4 Reading

R for Data Science (Wickham and Grolemund 2017):

- Chapter 12 *Tidy data*

R Programming for Data Science (Roger D. Peng 2020):

- Chapter 13 *Control structures*
- Chapter 14 *Functions*
- Chapter 17 *Regular expressions*

Check your understanding

In the regular expression "\d{4}", why are there two backslashes?

This is a peculiarity of using regular expressions in R. The first backslash tells R to treat the second backslash as a 'backslash' and not a special character.

What pattern would "\D{4}" match?

This would match any four characters that are NOT digits. (See also w vs W - i.e. lower-case w vs upper-case W).

In the world of regular expressions, what does “greedy” mean?

A greedy expression matches the longest possible pattern it can find in the target. A lazy expression takes the first matching pattern it comes across (i.e. the shortest). For example, here we try extracting 2 or more digits after the decimal point:

```
# Greedy...
str_extract("The value of pi is 3.14159", "\\d\\.\\d{2,}")
```

```
[1] "3.14159"
```

```
# Lazy (note the question mark)...
str_extract("The value of pi is 3.14159", "\\d\\.\\d{2,}?")
```

```
[1] "3.14"
```

5 Joining tables

5.1 Overview

This week we'll be looking at how to combine different tables of data.

By the end of this chapter you'll be able to:

1. Joint data frames using `dplyr::bind_rows()` or `dplyr::bind_cols()`
2. Describe and use the following `dplyr` mutating joins:

- `left_join()`
- `right_join()`
- `inner_join()`
- `full_join()`

3. Describe and use the following `dplyr` filtering joins:

- `semi_join()`
- `anti_join()`

5.2 Joining Tables

It's a common problem to have to combine data contained in two different tables. (We'll always assume in R that we are referring to data frames when we talk about a table - although there are other table-like structures)

5.2.1 Binding rows

One simple case is when we have two tables with identical columns and we want to "stick" one onto the bottom of the other. In Excel this is often done by cutting and pasting, which is a very dangerous method since it's so easy to lose rows.

In R we do this using `dplyr::bind_rows()`. (This is the tidyverse equivalent of `rbind()`.)

```
library(dplyr)
library(stringr)
# Split up mtcars
all_cars <- mtcars %>% arrange(wt)
```

```
big_cars <- filter(all_cars, wt > 3.5 )
little_cars <- filter(all_cars, wt <= 3.5)

# Recombine
combined_rows <- bind_rows(big_cars, little_cars) %>% arrange(wt)

# Test for equality
identical(all_cars, combined_rows)
```

```
[1] TRUE
```

A few things to note:

1. If one of the data frames has a column that isn't in the other, it will be filled with NAs in the output;
2. The columns don't have to be in the same order, they just have to have the same names;
3. If there are are duplicated rows in the inputs, they will be duplicated in the output (see `union()` for a way to avoid this if both data frames have identical columns);
4. `bind_rows()` can take any number of inputs - contained in a list;
5. If you are trying to join many data frames (inside a loop for example) it is much faster to collect them in a list and then bind them all at once, than it is to bind each one inside the loop.
6. You can use the `.id` parameter to record the source of each row (see `?bind_rows`).
7. The set functions `union()`, `intersect()` and `setdif()` can also be useful.

5.2.2 Binding columns

If we have the same number of rows in two data frames we can add the columns using `bind_cols()`.

```
# Split up mtcars
cars1 <- all_cars %>% select(1:4)
cars2 <- all_cars %>% select(-(1:4))

#Recombine
combined_cols <- bind_cols(cars1, cars2)

# Test for equality
identical(all_cars, combined_cols)
```

```
[1] TRUE
```

5.3 Mutating Joins

The following joins allow us to combine the data from two tables, creating extra columns as necessary.

A nice feature of these 'join_ functions is that their names and behaviour are similar to analogous functions for joining data in SQL, as we will see later in the programme.

Suppose I have a table containing some details of students registered on a particular module. Perhaps it contains the student ID and a mark for a particular assignment.

```
set.seed(123)
marks <-
  tibble(ID = 1:5,
         Score = round(rnorm(5, mean = 65, sd = 10), 1))

marks$ID[5] <- 25
marks
```

```
# A tibble: 5 x 2
     ID Score
  <dbl> <dbl>
1     1  59.4
2     2  62.7
3     3  80.6
4     4  65.7
5    25  66.3
```

I also have a table listing students and whether they are UG or PGT and their email address.

```
details <-
  tibble( ID = 1:10,
          Level = sample(c("UG", "PGT"), size = 10, replace = TRUE)) %>%
  mutate(Email = str_c("stu", ID, "@univ.ac.uk"))

is.na(details$Email) <- 3

details
```

```
# A tibble: 10 x 3
      ID Level Email
   <int> <chr> <chr>
 1     1 PGT   stu1@univ.ac.uk
 2     2 PGT   stu2@univ.ac.uk
```

```
 3     3 PGT   <NA>
 4     4 UG    stu4@univ.ac.uk
 5     5 PGT   stu5@univ.ac.uk
 6     6 UG    stu6@univ.ac.uk
 7     7 PGT   stu7@univ.ac.uk
 8     8 UG    stu8@univ.ac.uk
 9     9 UG    stu9@univ.ac.uk
10    10 UG    stu10@univ.ac.uk
```

Now suppose I want to add the email address to the first data frame. In Excel we could use something like `vlookup` or `index/match`. In R we can use a `left_join()` which takes each row in the second table and adds the columns from the second if it finds a match. In my experience this is the most commonly used join.

```
marks %>%
  left_join(details, by = "ID")
```

```
# A tibble: 5 x 4
     ID Score Level Email
  <dbl> <dbl> <chr> <chr>
1     1  59.4 PGT   stu1@univ.ac.uk
2     2  62.7 PGT   stu2@univ.ac.uk
3     3  80.6 PGT   <NA>
4     4  65.7 UG    stu4@univ.ac.uk
5    25  66.3 <NA>  <NA>
```

If you don't want the Level column it can be filtered out after the join.

`right_join()` includes all the rows in the second table.

```
marks %>%
  right_join(details, by = "ID")
```

```
# A tibble: 10 x 4
      ID Score Level Email
   <dbl> <dbl> <chr> <chr>
 1     1  59.4 PGT   stu1@univ.ac.uk
 2     2  62.7 PGT   stu2@univ.ac.uk
 3     3  80.6 PGT   <NA>
 4     4  65.7 UG    stu4@univ.ac.uk
 5     5  NA   PGT   stu5@univ.ac.uk
 6     6  NA   UG    stu6@univ.ac.uk
 7     7  NA   PGT   stu7@univ.ac.uk
 8     8  NA   UG    stu8@univ.ac.uk
```

```
 9     9  NA    UG    stu9@univ.ac.uk
10    10  NA    UG    stu10@univ.ac.uk
```

In this case we end up with 10 rows (as in the `details` table.)

`inner_join()` only matches rows that occur in both tables.

```
marks %>%
  inner_join(details, by = "ID")
```

```
# A tibble: 4 x 4
     ID Score Level Email
  <dbl> <dbl> <chr> <chr>
1     1  59.4 PGT   stu1@univ.ac.uk
2     2  62.7 PGT   stu2@univ.ac.uk
3     3  80.6 PGT   <NA>
4     4  65.7 UG    stu4@univ.ac.uk
```

Finally, `full_join()` includes all rows in `marks` OR `details`.

```
marks %>%
  full_join(details, by = "ID")
```

```
# A tibble: 11 x 4
      ID Score Level Email
   <dbl> <dbl> <chr> <chr>
 1     1  59.4 PGT   stu1@univ.ac.uk
 2     2  62.7 PGT   stu2@univ.ac.uk
 3     3  80.6 PGT   <NA>
 4     4  65.7 UG    stu4@univ.ac.uk
 5    25  66.3 <NA>  <NA>
 6     5  NA   PGT   stu5@univ.ac.uk
 7     6  NA   UG    stu6@univ.ac.uk
 8     7  NA   PGT   stu7@univ.ac.uk
 9     8  NA   UG    stu8@univ.ac.uk
10     9  NA   UG    stu9@univ.ac.uk
11    10  NA   UG    stu10@univ.ac.uk
```

Note that if the second table (`details`) contains more than one row that matches with ID in `marks` there will be more than one corresponding row in the output.

```
details <-
  details %>%
```

```
    bind_rows(tibble(ID = 4, Level = "PGT", Email = "stu4@univ.ac.uk"))
```

```
  marks %>%
    left_join(details, by = "ID")
```

```
# A tibble: 6 x 4
     ID Score Level Email
  <dbl> <dbl> <chr> <chr>
1     1  59.4 PGT   stu1@univ.ac.uk
2     2  62.7 PGT   stu2@univ.ac.uk
3     3  80.6 PGT   <NA>
4     4  65.7 UG    stu4@univ.ac.uk
5     4  65.7 PGT   stu4@univ.ac.uk
6    25  66.3 <NA>  <NA>
```

5.3.1 Filtering joins

Sometimes we want to remove rows from the first table, dependent on the contents of the second. For example we can use the information about student level in `details` to pick out any students with out a matching entry.

```
  marks %>%
    anti_join(details, by = "ID")
```

```
# A tibble: 1 x 2
     ID Score
  <dbl> <dbl>
1    25  66.3
```

Or we can use a list of UG students (in this case generated by filtering `details`) to pick out the UG students in marks.

```
  marks %>%
    semi_join(details %>% filter(Level =="UG"), by = "ID")
```

```
# A tibble: 1 x 2
     ID Score
  <dbl> <dbl>
1     4  65.7
```

5.4 Summary

Mutating joins add columns from y to x, matching rows based on the keys passed with the `by =` parameter:

`inner_join()` includes all rows in x and y.

`left_join()` includes all rows in x.

`right_join()` includes all rows in y.

`full_join()` includes all rows in x or y.

If a row in x matches multiple rows in y, all the rows in y will be returned once for each matching row in x.

Filtering joins filter rows from x based on the presence or absence of matches in y:

`semi_join()` returns all rows from x with a match in y.

`anti_join()` returns all rows from x without a match in y.

5.5 Reading

R for Data Science (Wickham and Grolemund 2017):

- *Chapter 13 Relational data*

Check your understanding

What would you expect the output from `full_join(big_cars, little_cars)` to be?

```
full_join(big_cars, little_cars)
```

```
Joining with `by = join_by(mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear,
carb)`
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
1	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
2	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
3	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
4	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
5	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
6	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
7	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2

```
8  16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
9  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
10 14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
11 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
12 30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
13 30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
14 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
15 27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
16 26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
17 32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
18 22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
19 21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
20 21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
21 19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
22 21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
23 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
24 22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
25 15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
26 24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
27 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
28 15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
29 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
30 19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
31 17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
32 18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
```

6 Clean and ethical data

6.1 Overview

This week we'll be looking at two topics:

- preparing less-than-perfect data sets for processing
- regulation and ethics of data science

6.2 Working with clean data

Remember the data science pipeline diagram from Week 1?

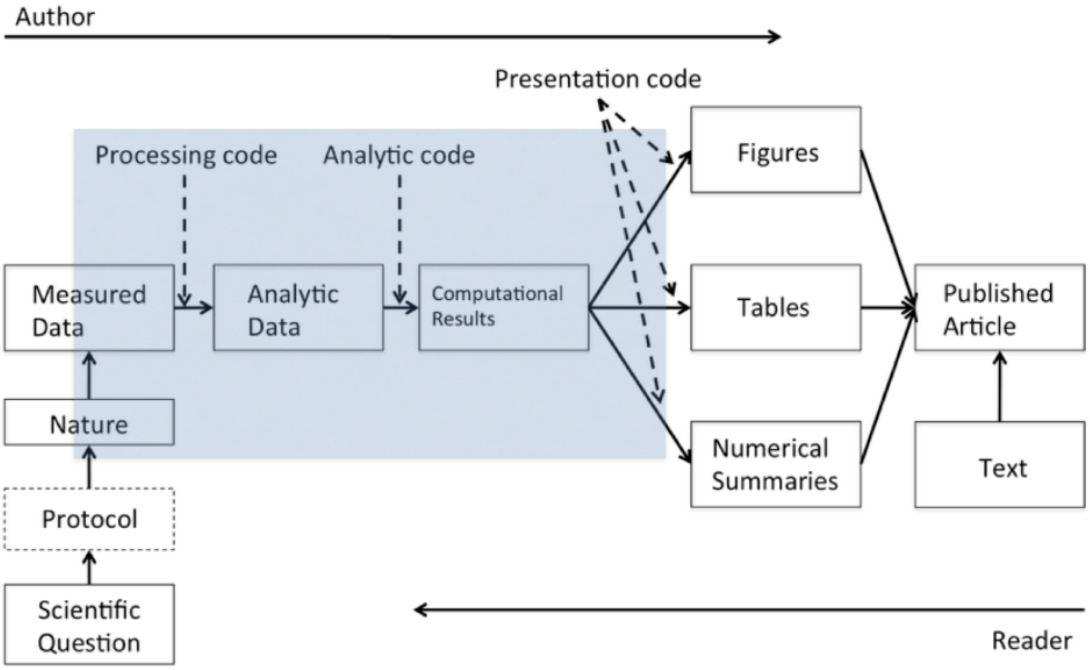


Figure 6.1: The Data Science Pipeline diagram

In this section we'll be focusing on the part labelled *processing code*.

The processing code carries out some, or all of the following functions:

1. getting the data
2. exploring the structure of the data
3. cleaning the data, so it is + technically correct + consistent
4. tidying the data

We've already looked at Stages 2 and 4. We've also covered some simple methods for Stage 1 - and we will look at more in future weeks. So in this section we'll be concentrating on Stage 3.

Technically correct data

We define technically correct data (Jonge and Loo 2013) from the point of view of the computer as a dumb machine:

- will the data actually load, or is it just rejected?
- is the data of the right type?
- is missing data correctly represented as NAs?
- are we using the right encoding for character elements?

Here are some useful rules:

- create headings with no spaces
  - use janitor
- don't use row names
  - use tibble::rownames_to_column
- dates in date format
  - use lubridate if you need to manipulate dates - but you don't always need to convert (e.g. if you are given a year as an integer, or you are just using the data as a label and the format is fine)
- remove columns you don't need
- if you have time series data, consider a time series package like `zoo`
  - not covered in this module
- Strike a balance between doing the minimum necessary to create a dataset that is "clean" enough for the job in hand and one that has every single issue fixed so it can be used without further processing for future tasks.
- Document what you've done

The language around quality control isn't always consistent in the way different words are used - but I would call this stage data *verification*.

Consistent data

The next stage is to make sure the data is *consistent*. This means: it is internally logically consistent (e.g. if one column can be calculated from others, it agrees with what you would expect); it is consistent with real-world logic; it is consistent with business logic and assumptions (there may be some overlap between those last two).

Properties of consistent data:

- it is consistent with the column heading
- it has a consistent format
- the same thing always has the same label
- it is logically consistent
  - e.g. date_of_birth < date_of_death
- it follows "business" rules and assumptions
  - e.g. a student id consists of eight digits
  - you may need to get a domain expert to give input here
  - beware of validation rules that may seem reasonable to you but are actually too restrictive
- missing, or invalid data is dealt with appropriately

With many data sets, the cleaning stage is one of the most time consuming parts of the whole process. In deciding the approach to take, and when the output if good enough, you should think carefully about the requirements of the job and the needs of the end-user. Sometimes you will need 100% correct data and sometimes you won't. You should also be aware of any regulations - the GDPR (see below) has lots to say about data quality.

Checking data for consistency is known as data *validation* . (Compare with the ability to include data validation rules in Excel.) There is a package called `validate` (Loo and Jonge 2021) that can help.

Deciding what to do about missing data is often a tricky but crucial decision. Three options are:

- Remove or ignore records with missing data
  - be careful when doing this, especially if you decide to remove a whole row (record) because of a single missing column (field)
- Try to find another source of information for the missing data
- Fill in the missing items with an estimated value (such as the mean, median or a different modelled value).

The third option is known as imputation and there is a whole page on CRAN listing R packages in the Missing Data Task View.

6.2.1 A hierarchy for data cleaning

1. corrections that can be universally applied
2. corrections that can be applied to a group identified by a business rule
3. ad hoc corrections to single items

Changes should usually be applied in this order - but remember, it's an iterative process.

Always make changes programmatically and document your reasons.

6.3 Data regulation

The UK data protection regime is set out in the Data Protection Act 2018 (DPA) and the General Data Protection Regulation (GDPR) which also forms part of UK law.

The source for most of this section is the UK Information Commissioner's Office but the principles apply everywhere.

6.3.1 What is Data Protection?

Data protection is about ensuring people can trust you to use their data fairly and responsibly.

From a 2019 survey:

Nearly one in three (32%) people have high trust and confidence in companies and organisations storing and using their personal information, which is slightly down from the 34% stating this in 2018.

The proportion stating 'none at all' has marginally increased from 9% to 10%.

Source: harris interactive survey for the Information Commissioner's Office

Data protection is the fair and proper use of information about people. It's part of the fundamental right to privacy but on a more practical level, it's really about building trust between people and organisations. It's about treating people fairly and openly, recognising their right to have control over their own identity and their interactions with others, and striking a balance with the wider interests of society.

It's also about removing unnecessary barriers to trade and co-operation.

Data protection (trust) is essential to innovation.

The Data Protection Act covers **personal information**, which means information about a particular living individual.

It doesn't need to be 'private' information: even information which is public knowledge or is about someone's professional life can be personal data.

It doesn't cover truly anonymous information but if you could still identify someone from the details, or by combining it with other information, it will still count as personal data.

6.3.2 The GDPR sets out seven key principles:

- Lawfulness, fairness and transparency
- Purpose limitation
- Data minimisation
- Accuracy
- Storage limitation
- Integrity and confidentiality (security)
- Accountability

Lawfulness, fairness and transparency

- You must identify valid grounds under the GDPR (known as a 'lawful basis') for collecting and using personal data.
- You must ensure that you do not do anything with the data in breach of any other laws.
- You must use personal data in a way that is fair. This means you must not process the data in a way that is unduly detrimental, unexpected or misleading to the individuals concerned.
- You must be clear, open and honest with people from the start about how you will use their personal data.

Purpose limitation

- You must be clear about what your purposes for processing are from the start.
- You need to record your purposes as part of your documentation obligations and specify them in your privacy information for individuals.
- You can only use the personal data for a new purpose if either this is compatible with your original purpose, you get consent, or you have a clear obligation or function set out in law.

Data minimisation

You must ensure the personal data you are processing is:

- adequate sufficient to properly fulfil your stated purpose;
- relevant has a rational link to that purpose; and
- limited to what is necessary - you do not hold more than you need for that purpose.

Accuracy

- You should take all reasonable steps to ensure the personal data you hold is not incorrect or misleading as to any matter of fact.
- You may need to keep the personal data updated, although this will depend on what you are using it for.
- If you discover that personal data is incorrect or misleading, you must take reasonable steps to correct or erase it as soon as possible.
- You must carefully consider any challenges to the accuracy of personal data.

Storage limitation

- You must not keep personal data for longer than you need it.
- You need to think about and be able to justify how long you keep personal data. This will depend on your purposes for holding the data.
- You need a policy setting standard retention periods wherever possible, to comply with documentation requirements.
- You should also periodically review the data you hold, and erase or anonymise it when you no longer need it.
- You must carefully consider any challenges to your retention of data.
- Individuals have a right to erasure if you no longer need the data.
- You can keep personal data for longer if you are only keeping it for public interest archiving, scientific or historical research, or statistical purposes.

Integrity and confidentiality (security)

- You must ensure that you have appropriate security measures in place to protect the personal data you hold.

Accountability

- The accountability principle requires you to take responsibility for what you do with personal data and how you comply with the other principles.
- You must have appropriate measures and records in place to be able to demonstrate your compliance.

6.4 Ethical Standards

6.4.1 The IFoA and RSS Guide for Ethical Data Science

The Institute and Faculty of Actuaries and the Royal Statistical Society have jointly produced a Guide for Ethical Data Science (IFoA and RSS 2019) which is quoted below.

The Guide has five themes:

- Seek to enhance the value of data science for society
- Avoid harm
- Apply and maintain professional competence
- Seek to preserve or increase trustworthiness
- Maintain accountability and oversight

Seek to enhance the value of data science for society

As the impact that data science can have on society could be significant, an important ethical consideration is what the potential implications could be on society as a whole.

A common theme within ethical frameworks discussing data science and AI is for practitioners to attempt to seek outcomes within their work which support the improvement of public wellbeing. This could involve practitioners seeking to share the benefits of data science and balancing this with the wellbeing of potentially affected individuals.

Avoid harm

Data science has the potential to cause harm and this ethical consideration therefore focuses on how practitioners can avoid this by working in a manner that respects the privacy, equality and autonomy of individuals and groups, and speaking up about potential harm or ethical violations.

Practitioners may be subject to legal and regulatory obligations in relation to the privacy of individuals, relevant to the jurisdiction in which they are working, as well as regulatory obligations to speak up about harm or violations of legal requirements.

This can also be applied to work relating to businesses, animals or the environment, with consideration of commercial rights, animal welfare and the protection of environmental resources.

The recent EU paper on profiling and automatic decision making is of interest in this context.

Apply and maintain professional competence

This ethical principle expects data science practitioners to apply best practice and comply with all relevant legal and regulatory requirements, as well as applicable professional body codes.

Professional competence involves fully understanding the sources of error and bias in data, using 'clean' data (eg edited for missing, inconsistent or erroneous values), and supporting work with robust statistical and algorithmic methods that are appropriate to the question being asked.

Practitioners can also thoroughly assess and balance the benefits of the work versus the risks posed by it, and keep models under regular review.

Seek to preserve or increase trustworthiness

The public's trust and confidence in the work of data scientists can be affected by the way ethical principles are applied. Practitioners can help to increase the trustworthiness of their work by considering ethical principles throughout all stages of a project.

This is another reoccurring theme that encourages practitioners to be transparent and honest when communicating about the way data is used. Transparency can include fully explaining how algorithms are being used, if and why any decisions have been delegated, and being open about the risks and biases.

Engaging widely with a diverse range of stakeholders and considering public perceptions both from the outset, and throughout projects, can help to build trustworthiness and ensure all potential biases are understood.

Maintain accountability and oversight

Another key issue in data ethics around automation and AI is the question of how practitioners maintain human accountability and oversight within their work.

Being accountable can include being mindful of how and when to delegate any decision making to systems, and having governance in place to ensure systems deliver the intended objectives.

When deciding to delegate any decision making, it would be useful to fully understand and explain the potential implications of doing so, as the work could lead to introducing advanced AI systems which do not have adequate governance. Practitioners should note that delegating any decisions to these systems does not remove any of their individual responsibilities.

6.4.2 Other ethics frameworks

You should research any other ethical frameworks relevant to the jurisdiction and professional area you're working in.

For example, the UK Government has published a Data Ethics Framework.

And here is a link to the IEEE's work on the ethics of autonomous and intelligent systems.

7 Some other data structures

7.1 Overview

This week we'll be covering different data structures:

- XML
- JSON
- unstructured (or semi-structured) text

7.2 XML

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

XML can serve as the basis for defining markup languages for particular domains. For example XBRL (Extensible Business Reporting Language), KML (Keyhole Markup Language for geographic information), BeerXML (you guessed it).

Here is a simple example containing some data about my pets.

```
<pets>
  <pet id = '001' species = 'dog'>
    <tag>Rover</tag>
    <colour>black</colour>
  </pet>
  <pet id = '002' species = 'cat'>
    <tag>Tiddles</tag>
    <colour>ginger</colour>
  </pet>
  <pet id = '003' species = 'dog'>
    <tag>Fido</tag>
    <colour>brownish</colour>
  </pet>
</pets>
```

Even if you knew nothing about XML before, you can work out what is going on.

7.2.1 Package `xml2`

The package `xml2` gives you tools to read an XML file and extract the data.

https://blog.rstudio.com/2015/04/21/xml2/

Normally you would read the data from an external file, but for a very small example I've saved the data above as a string called `my_pets`.

7.2.2 Navigating the tree

In XML everything is arranged in a tree structure, and each element is called a **node**.

You are familiar with a tree structure - just think of the way the files are organised on your computer: folders sit inside folders and each folder can contain other folders and files.

In a similar way, nodes sit inside nodes and each node can contain other nodes. The first node, that contains everything else, is the root node, and a node that doesn't contain any other nodes is called a leaf node.

In our example … is the root node, and the text strings "Fido", "Brownish" etc are leaf nodes.

Nodes can also contain **attributes**. For example the … nodes have "id" and "species" attributes.

Every node, except the root, has exactly one parent, and nodes can have children and siblings (nodes with the same parent).

We can use these concepts to navigate the tree.

```
xpets <- read_xml(my_pets)
xml_name(xpets)  # The name of the root node
```

```
[1] "pets"
```

```
xml_child(xpets) # Finds the first child of the root (Rover)
```

```
{xml_node}
<pet id="001" species="dog">
[1] <tag>Rover</tag>
[2] <colour>black</colour>
```

```
xml_children(xpets) # Finds all the children of the root (Rover, Fido & Tibbles)
```

```
{xml_nodeset (3)}
[1] <pet id="001" species="dog">\n  <tag>Rover</tag>\n  <colour>black</colour ...
[2] <pet id="002" species="cat">\n  <tag>Tiddles</tag>\n  <colour>ginger</col ...
[3] <pet id="003" species="dog">\n  <tag>Fido</tag>\n  <colour>brownish</colo ...
```

```r
xml_children(xpets) |> xml_name() # The name of each child
```

```
[1] "pet" "pet" "pet"
```

```r
xml_child(xpets) |> xml_siblings() # the siblings of Rover
```

```
{xml_nodeset (2)}
[1] <pet id="002" species="cat">\n  <tag>Tiddles</tag>\n  <colour>ginger</col ...
[2] <pet id="003" species="dog">\n  <tag>Fido</tag>\n  <colour>brownish</colo ...
```

```r
xml_child(xpets) |> xml_parent() # The parent of Rover
```

```
{xml_node}
<pets>
[1] <pet id="001" species="dog">\n  <tag>Rover</tag>\n  <colour>black</colour ...
[2] <pet id="002" species="cat">\n  <tag>Tiddles</tag>\n  <colour>ginger</col ...
[3] <pet id="003" species="dog">\n  <tag>Fido</tag>\n  <colour>brownish</colo ...
```

```r
xml_child(xpets) |> xml_child() # <tag> and <colour> are children of Rover
```

```
{xml_node}
<tag>
```

7.2.3 Searching

We can navigate and search through the tree with more precision using XPath.

```r
xml_find_first(xpets, '//pet[@species="cat"]') # Find the first cat
```

```
{xml_node}
<pet id="002" species="cat">
[1] <tag>Tiddles</tag>
[2] <colour>ginger</colour>
```

```
xpets %>%
      xml_find_all("//pet[@species='dog']") %>% # List the dogs' names
      xml_find_all(".//tag") %>%                # Note the important .
      xml_text()
```

```
[1] "Rover" "Fido"
```

```
xpets %>%
  xml_find_all(".//pet[@species='dog']") %>%
  xml_attr("id")
```

```
[1] "001" "003"
```

7.2.4 Create a data frame

By extracting each quantity we want separately we can put together a data frame.

```
pet_id <-
xpets %>%
  xml_find_all(".//pet") %>%
  xml_attr("id")
```

```
pet_species <-
xpets %>%
  xml_find_all(".//pet") %>%
  xml_attr("species")
```

```
pet_name <-
  xpets %>%
  xml_find_all(".//pet/tag") %>%
  xml_text()
```

```
pet_colour <-
  xpets %>%
  xml_find_all(".//pet/colour") %>%
  xml_text()
```

```
dfpets <-
  data.frame(ID = pet_id,
             Name = pet_name,
             Species = pet_species,
```

```
              Colour = pet_colour)
dfpets
```

```
   ID    Name Species   Colour
1 001   Rover     dog    black
2 002 Tiddles     cat   ginger
3 003    Fido     dog brownish
```

7.3 JSON

JSON is a syntax for storing and exchanging data. It's lightweight, human readable, language-independent and very widely used. Most programming languages can process JSON.

To see what JSON looks like we'll use the `jsonlite` package to convert our pets data from a data frame to JSON.

```
library(jsonlite)
jpets <- toJSON(dfpets)
jpets
```

```
[{"ID":"001","Name":"Rover","Species":"dog","Colour":"black"},{"ID":"002","Name":"Tiddles"
```

JSON stands for **JavaScript Object Notation**. The text inside each set of curly brackets represents a JavaScript object - but it is just text and can be taken to JSON is independent of JavaScript. You can think of each object as being like a card in an old-fashioned card index system.

We can make it easier to read:

```
prettify((jpets))
```

```
[
    {
        "ID": "001",
        "Name": "Rover",
        "Species": "dog",
        "Colour": "black"
    },
    {
        "ID": "002",
        "Name": "Tiddles",
```

```
        "Species": "cat",
        "Colour": "ginger"
    },
    {
        "ID": "003",
        "Name": "Fido",
        "Species": "dog",
        "Colour": "brownish"
    }
]
```

`jsonlite`has a function to convert to a data frame.

```
fromJSON(jpets)
```

```
  ID    Name Species   Colour
1 001   Rover     dog    black
2 002 Tiddles     cat   ginger
3 003    Fido     dog brownish
```

7.4 Text processing

Text processing (a branch of Natural Language Processing, or NLP) is a big topic and we can only scratch the surface in this module.

In Wednesday's class we will look at an example of sentiment analysis.

We'll use a list of English words rated for valence with an integer between minus five (negative) and plus five (positive). The words have been manually labeled by Finn Årup Nielsen in 2009-2011.(Nielsen 2011) but I have removed profanities from the list because I used it for a presentation in a school.

We'll also use some classic texts downloaded from Project Gutenburg.

Please see the Further Reading for more information on text processing (and you'll probably want to refresh your regex skills).

7.5 Reading

- XML
  - What is XML (Just the first bit)
  - Parse and process XML (and HTML) with xml2

- JSON
  - Intro to JSON

7.6 Further reading

Text Mining with R: a tidy approach (Julia and David 2017) is an excellent introduction, compatible with the methods used in this course.

Check your understanding

Convert the `starwars` data to JSON and examine the structure.

```
jstarwars <- toJSON(starwars, pretty = TRUE)
```

Try converting back to a data frame and compare with the original.

```
dfstarwars <- fromJSON(jstarwars)

identical(starwars, dfstarwars)
```

```
[1] FALSE
```

`identical()` returns FALSE. Can you work out why?

8 SQL

8.1 Overview

This week we'll be looking at the database language SQL.

You can't really be a data scientist without knowing something about SQL - it's the language used to interact with most of the databases (small and very large) in the world.

In the task for this week you will use the `RSQLite`(Müller et al. 2023) package to create a local database (using SQLite) which you will then use as a source to answer various queries

Start by skimming through the first few sections of the w3schools tutorial listed in the reading.

Have a quick look at the documentation for `RSQLite` (also referenced in the reading).

Then, open the file `QueryDemos.Rmd` and get working!

For interest, there's information about connecting to external databases here: https://db.rstudio.com/ .

8.2 Other types of database.

SQL is designed to work with so-called *relational databases*. These are the most common sort of database, where the data is store in tables, much like data frames in concept. There is both strong theoretical backing and decades of technical development to support relational databases but there are other possibilities and, as always, the best solution depends on the task and the context.

Alternatives to relational databases are often called NoSQL (where the "No" stands for "Not only". The topic is a bit beyond the scope of this module but it's an interesting and important one, so I thought I'd give you some brief pointers and I encourage you to do your own reading if you're interested.

The main types of NoSQL data bases are:

- key-value
  - a key-value database (or "store") is a set of unique identifiers (the keys) each of which has an associated value. You'er probably familiar with this concept, for example we saw it when we were looking at JSON.

- column-family (or column-oriented)
  - column-family databases are optimised for aggregating functions (down columns) like sum, maximum, etc. So they can be useful for applications where analytics are important.
- document databases
  - a document database is a type of key-value database where the "value" is a document with its own internal structure that can be queried. Probably the best known supplier is MongoDB
- graph databases
  - in a graph database data is stored in the form of a graph (the mathematical type of graph consisting of nodes and edges, not a bar chart!). The edges, or links, between nodes carry semantic information, for example `(Nigel) -- teaches -- (statistics)`. Here, `(Nigel)` and `(statistics)` are nodes and the link between them expresses the relationship.

8.3 Reading

- https://www.w3schools.com/sql/sql_intro.asp
- https://db.rstudio.com/databases/sqlite/
- https://www.datacamp.com/courses/intro-to-sql-for-data-science (if you are interested)

9 APIs

9.1 Overview

This week we'll be looking at APIs

9.2 Definitions

9.2.1 API

An application programming interface (API) is an interface or communication protocol between different parts of a computer program intended to simplify the implementation and maintenance of software. (Wikipedia)

9.2.2 REST

Representational state transfer (REST) is a software architectural style that defines a set of constraints to be used for creating Web services. (Wikipedia)

Specifically, one of the restful rules is that that you should get data (called a resource) returned when you link to a specific URL.

The URL is called a **request** and what is sent back is called a **response**.

You can use restful APIs to send as well as receive data, but we will only look at how to get data.

The API request can be included in a program - so you don't need a user to click on a *download* link.

Another piece of jargon is **endpoint**. This is the base url for the API. This is followed by a **path** that points to the exact resource.

Finally we can have **query parameters**. These always begin with a ? and look like:

`?query1=param1&query2=param2`

where the & separates two query/parameter pairs.

Let's have an example.

9.3 Example

The endpoint for Github is: `https://api.github.com`

The path to a specific user's repos is `/users/<username>/repos`.

Try copying `https://api.github.com/users/vivait/repos` into your browser…

you should see information returned in JSON.

But we want to access the data in a program, not via a browser.

The package `httr` provides tools for HTTP, including the verb GET:

```
library(dplyr)
library(jsonlite)
library(httr)
```

```
github_api <- function(path) {
  url <- modify_url("https://api.github.com", path = path)
  GET(url)
}

resp <- github_api("/users/actuarial-science/repos")
```

We can use `jsonlite` to parse the content of the response into a useful R object.

```
repos <- fromJSON(content(resp, "text"))
```

We can add some parameters to our query

```
resp <- github_api("/users/vivait/repos?sort=updated&per_page=100")
repos <- fromJSON(content(resp, "text"))
```

In fact, if we know the request will return JSON, we can parse it directly with `jsonlite`. (Not advised in a program.)

For example, the Github documentation says *You can issue a GET request to the root endpoint to get all the endpoint categories that the REST API v3 supports*:

```
head(fromJSON("https://api.github.com"), 10)
```

```
$current_user_url
[1] "https://api.github.com/user"

$current_user_authorizations_html_url
[1] "https://github.com/settings/connections/applications{/client_id}"
```

```
$authorizations_url
[1] "https://api.github.com/authorizations"

$code_search_url
[1] "https://api.github.com/search/code?q={query}{&page,per_page,sort,order}"

$commit_search_url
[1] "https://api.github.com/search/commits?q={query}{&page,per_page,sort,order}"

$emails_url
[1] "https://api.github.com/user/emails"

$emojis_url
[1] "https://api.github.com/emojis"

$events_url
[1] "https://api.github.com/events"

$feeds_url
[1] "https://api.github.com/feeds"

$followers_url
[1] "https://api.github.com/user/followers"
```

9.4 Twitter example

NOTE the Twitter (X) API examples below, no longer work (thanks Elon)

They will be replaced soon.

This code demonstrates how to use the `rtweet` package.

For more detail, see https://cran.r-project.org/web/packages/rtweet/vignettes/intro.html.

First you'll need to set up a developer account with Twitter and get the access keys you need by creating a new app.

Follow the instructions at: https://cran.r-project.org/web/packages/rtweet/vignettes/auth.html.

```
# library(rtweet)
# ## authenticate - insert your app name and keys below
# token <- create_token(
#   app = "R camlad",
```

```
#   consumer_key = api_key,
#   consumer_secret = api_secret_key,
#   access_token = access_token,
#   access_secret = access_token_secret)
```

9.4.1 Following a hashtag

We can search for tweets including a particular hashtag.

```
## search for tweets using the Cardano hashtag
# rt <- search_tweets("#Cardano", n = 100, include_rts = FALSE)
#
# ## preview tweets data
# rt %>% select(id, text)
```

9.4.2 Trending in Leicester

```
# trnds <- get_trends("Leicester")
# trnds %>%
#   select(trend, tweet_volume) %>%
#   arrange(desc(tweet_volume))
```

9.4.3 Get a particular user's timeline

```
library(stringr)
# tmls <- get_timeline("leicspolice", n = 100)
#
# tmls %>%
#   select(created_at, text) %>%
#   filter(str_detect(text, 'Traffic'))
```

9.5 Accessing UK census (and other) data

Our final example demonstrates the NOMIS API, which can be accessed through the `nomisr`(Odell 2018) package.

9.5.1 A quick demonstration of using `nomisr` to extract data from the Nomis API

This example is based on the nomisr introduction vignette

```
library(nomisr)
```

First, we can download information on what data is available.

```
data_info <- nomis_data_info()
#head(data_info)
glimpse(data_info)
```

```
Rows: 1,605
Columns: 14
$ agencyid                                <chr> "NOMIS", "NOMIS", "NOMIS", "NOMIS~
$ id                                      <chr> "NM_1_1", "NM_2_1", "NM_4_1", "NM~
$ uri                                     <chr> "Nm-1d1", "Nm-2d1", "Nm-4d1", "Nm~
$ version                                 <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
$ annotations.annotation                  <list> [<data.frame[10 x 2]>], [<data.f~
$ components.attribute                    <list> [<data.frame[7 x 4]>], [<data.fr~
$ components.dimension                    <list> [<data.frame[5 x 3]>], [<data.fr~
$ components.primarymeasure.conceptref    <chr> "OBS_VALUE", "OBS_VALUE", "OBS_VA~
$ components.timedimension.codelist       <chr> "CL_1_1_TIME", "CL_2_1_TIME", "CL~
$ components.timedimension.conceptref     <chr> "TIME", "TIME", "TIME", "TIME", "~
$ description.value                       <chr> "Records the number of people cla~
$ description.lang                        <chr> "en", "en", "en", "en", "en", "en~
$ name.value                              <chr> "Jobseeker's Allowance with rates~
$ name.lang                               <chr> "en", "en", "en", "en", "en", "en~
```

There's a lot here (`data_info` has 1605 rows). To dig deeper we can search the column `description.value` or `name.value` for key words.

```
pop_data_info <-
  data_info %>%
  filter(str_detect(name.value, "(?i)population")) %>%
  select(id, name.value)

#pop_data_info %>% head()
glimpse(pop_data_info)
```

```
Rows: 110
Columns: 2
$ id         <chr> "NM_17_1", "NM_17_5", "NM_31_1", "NM_100_1", "NM_136_1", "N~
```

```
$ name.value <chr> "annual population survey", "annual population survey (vari~
```

Suppose we wanted population data for Leicester. It looks like "NM_31_1" might be worth investigating, so we can dig down deeper.

The data or is categorised first by "concept" (Read the docs at nomis if you want more details.)

```
id = "NM_31_1"
nomis_get_metadata(id)
```

```
# A tibble: 6 x 3
  codelist          conceptref isfrequencydimension
  <chr>             <chr>      <chr>
1 CL_31_1_GEOGRAPHY GEOGRAPHY  false
2 CL_31_1_SEX       SEX        false
3 CL_31_1_AGE       AGE        false
4 CL_31_1_MEASURES  MEASURES   false
5 CL_31_1_FREQ      FREQ       true
6 CL_31_1_TIME      TIME       false
```

GEOGRAPHY looks relevant, so we explore what "types" are available.

```
nomis_get_metadata(id, "GEOGRAPHY", type = "type")
```

```
# A tibble: 26 x 3
   id      label.en                                                description.en
   <chr>   <chr>                                                   <chr>
 1 TYPE83  jobcentre plus group as of April 2019                   jobcentre plu~
 2 TYPE84  jobcentre plus district as of April 2019                jobcentre plu~
 3 TYPE342 english index of multiple deprivation 2010 - deciles    english index~
 4 TYPE347 scottish index of multiple deprivation 2009 - deciles   scottish inde~
 5 TYPE349 welsh index of multiple deprivation 2008 - deciles      welsh index o~
 6 TYPE431 local authorities: county / unitary (as of April 2021) local authori~
 7 TYPE432 local authorities: district / unitary (as of April 20~ local authori~
 8 TYPE433 local authorities: county / unitary (as of April 2019) local authori~
 9 TYPE434 local authorities: district / unitary (as of April 20~ local authori~
10 TYPE442 combined authorities                                   combined auth~
# i 16 more rows
```

Finally, we can choose a particular type and investigate it.

```
  id %>%
    nomis_get_metadata("GEOGRAPHY", type = "TYPE446") %>%
    filter(str_detect(label.en, "Leicester"))
```

```
# A tibble: 2 x 4
  id         parentCode label.en       description.en
  <chr>      <chr>      <chr>          <chr>
1 1870659636 2013265924 Leicester      Leicester
2 1870659640 2013265924 Leicestershire Leicestershire
```

Looks like we've found what we want!

```
  leics_pop <-
    nomis_get_data(id = id, time = "latest",
                   geography = c("1870659636", "1870659640"))

  leics_pop %>%
    select(DATE, GEOGRAPHY_NAME, SEX_NAME, AGE_NAME, MEASURES_NAME, OBS_VALUE) %>%
    head(10)
```

```
# A tibble: 10 x 6
    DATE GEOGRAPHY_NAME SEX_NAME AGE_NAME           MEASURES_NAME OBS_VALUE
   <dbl> <chr>          <chr>    <chr>              <chr>              <dbl>
 1  2021 Leicester      Male     All ages           Value                 NA
 2  2021 Leicester      Male     All ages           Percent               NA
 3  2021 Leicester      Male     Aged under 1 year  Value                 NA
 4  2021 Leicester      Male     Aged under 1 year  Percent               NA
 5  2021 Leicester      Male     Aged 1 - 4 years   Value                 NA
 6  2021 Leicester      Male     Aged 1 - 4 years   Percent               NA
 7  2021 Leicester      Male     Aged 5 - 9 years   Value                 NA
 8  2021 Leicester      Male     Aged 5 - 9 years   Percent               NA
 9  2021 Leicester      Male     Aged 10 - 14 years Value                 NA
10  2021 Leicester      Male     Aged 10 - 14 years Percent               NA
```

9.6 Homework

Install the package `randNames` and, using the instructions in the package documentation register for a free API key at randomapi.com.

Write a programme to download random data for 400 imaginary users. What is the distribution of genders and country of origin in this data.

9.6.1 Optional Christmas Bonus question

Register an account at Advent of Code. For the **2020** competition solve Question 2. (The key to solving this elegantly is reading the data in and wrangling it into the best format to solve the problem.)

10 Final reports

10.1 Overview

In Week 10 you will be submitting your final reports.

10.2 Citing R and packages

The underlying programme is clearly a massive piece of work and you should give its authors credit by citing them in any work you do using R.

For example you might write:

> Statistical analysis was done using R 4.1.1 (R Core Team 2022).

R makes it easy to generate the right reference to use because there's a built-in function to do it.

```
citation()
```

```
To cite R in publications use:

  R Core Team (2022). R: A language and environment for statistical
  computing. R Foundation for Statistical Computing, Vienna, Austria.
  URL https://www.R-project.org/.

A BibTeX entry for LaTeX users is

  @Manual{,
    title = {R: A Language and Environment for Statistical Computing},
    author = {{R Core Team}},
    organization = {R Foundation for Statistical Computing},
    address = {Vienna, Austria},
    year = {2022},
    url = {https://www.R-project.org/},
  }
```

```
We have invested a lot of time and effort in creating R, please cite it
when using it for data analysis. See also 'citation("pkgname")' for
citing R packages.
```

You should also cite the main packages you've used. For example:

> Data wrangling was carried out with `dplyr` (Wickham et al. 2023) and other packages from the `tidyverse` (Wickham et al. 2019); graphs were plotted using `ggplot2` (Wickham 2016).

Again, you can use `citation` to generate the correct references. For example:

```
citation('dplyr')
```

```
To cite package 'dplyr' in publications use:

  Wickham H, François R, Henry L, Müller K, Vaughan D (2023). _dplyr: A
  Grammar of Data Manipulation_. R package version 1.1.3,
  <https://CRAN.R-project.org/package=dplyr>.

A BibTeX entry for LaTeX users is

  @Manual{,
    title = {dplyr: A Grammar of Data Manipulation},
    author = {Hadley Wickham and Romain François and Lionel Henry and Kirill Müller and Da
    year = {2023},
    note = {R package version 1.1.3},
    url = {https://CRAN.R-project.org/package=dplyr},
  }
```

If you are wondering, `BibTeX` is a file format (and software) used to describe lists of references, often for use within LaTeX documents. The `rmarkdown` package (Xie, Allaire, and Grolemund 2018) provides methods to work with BibTeX references.

10.2.1 What to cite?

You should always cite R itself, but there is an element of judgement in deciding which individual packages to cite. For this module, you won't lose marks as long as you have made a reasonable effort.

To make your work fully reproducible it is best practice to list all the packages you've used (e.g. in an appendix). You can generate the necessary information with the function `sessionInfo()`, but this is not necessary for FDS coursework.

References

Chang, Winston. 2020. *R Graphics Cookbook*. O'Reilly Media. https://r-graphics.org/.

Friedl, Jeffrey E. F. 2006. *Mastering Regular Expressions*. 3rd ed.. Sebastapol, Calif.: O'Reilly.

IFoA, and RSS. 2019. "A Guide to Ethical Data Science." Institute; Faculty of Actuaries; Royal Statistical Society. https://www.actuaries.org.uk/system/files/field/document/An%20Ethical%20Charter%20for%20Date%20Science%20WEB%20FINAL.PDF.

Jonge, E. de, and M. van der Loo. 2013. "An Introduction to Data Cleaning with r." Discussion Paper / Statistics Netherlands. Statistics Netherlands. https://cran.r-project.org/doc/contrib/de_Jonge+van_der_Loo-Introduction_to_data_cleaning_with_R.pdf.

Julia, PhD Silge, and PhD Robinson David. 2017. *Text Mining with r: A Tidy Approach*. O'Reilly Media. https://www.tidytextmining.com/index.html.

Loo, Mark P. J. van der, and Edwin de Jonge. 2021. "Data Validation Infrastructure for r." *Journal of Statistical Software* 97 (10): 1–31. https://doi.org/10.18637/jss.v097.i10.

Müller, Kirill, Hadley Wickham, David A. James, and Seth Falcon. 2023. *RSQLite: 'SQLite' Interface for r*. https://CRAN.R-project.org/package=RSQLite.

Nielsen, F. Å. 2011. "AFINN." Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby: Informatics; Mathematical Modelling, Technical University of Denmark. http://www2.compute.dtu.dk/pubdb/pubs/6010-full.html.

Odell, Evan. 2018. "nomisr: Access Nomis UK Labour Market Data with r." *The Journal of Open Source Software* 3 (27): 859. https://doi.org/10.21105/joss.00859.

Peng, Roger D. 2020. *R Programming for Data Science*. Morrisville: Lulu.com. https://bookdown.org/rdpeng/rprogdatascience/.

Peng, Roger D. 2019. *Report Writing for Data Science in r*. British Columbia, Canada: Leanpub. https://leanpub.com/reportwriting.

Posit team. 2023. *RStudio: Integrated Development Environment for r*. Boston, MA: Posit Software, PBC. http://www.posit.co/.

R Core Team. 2022. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Robin Lovelace, Jannes Muenchow, Jakub Nowosad. 2020. *Geocomputation with r*. CRC Press. https://geocompr.robinlovelace.net/.

Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.

Wickham, Hadley, Romain François, Lionel Henry, Kirill Müller, and Davis Vaughan. 2023. *Dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dply

r.

Wickham, Hadley, and Garrett Grolemund. 2017. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly Media. http://r4ds.had.co.nz/.

Xie, Yihui, J. J. Allaire, and Garrett Grolemund. 2018. *R Markdown: The Definitive Guide*. Boca Raton, Florida: Chapman; Hall/CRC. https://bookdown.org/yihui/rmarkdown.