2025-11-04-strathclyde 1. Welcome to R notes

START THE SLIDES

1. Summary and Setup

- Whether you're on a university machine or your own laptop, you'll need to have both `R` and `RStudio` installed for this part of the course.
- **CHECK EVERYONE HAS ACCESS TO R/RSTUDIO**

Learning Objectives

Our goals in this course are:

- to introduce you to the fundamentals of `R` and `RStudio`, and of programming
- how to manage and process your data with an approach known as the `tidyverse`
- how to produce publication-quality data visualisations with the `ggplot2` package
- and how to combine documentation with analysis and code, using `RMarkdown`
- Working with a programming language (especially if it’s your first time) can be intimidating
    - But knowing even a little bit about `R` and how to manage and handle your own data is incredibly empowering
    - The rewards outweigh any frustrations.
- It's not a secret, but even experienced programmers find coding difficult and frustrating at times
    - I've been coding for 40 years and - though I know a bit more than when I started - every time a new language or technique turns up that I want to try, I'm a beginner all over again and feel like I know nothing.
    - So don't let intimidation stop you
    - With time and practice it just gets easier and easier to accomplish what you want.
- Learning to code opens up the full possibilities of computing for analysing your data and automating your work
    - Think of it this way: if you could only do biotechnology using kits you bought off Sigma, you could probably get a lot done.
    - But - if you don’t understand the biochemistry of the kit - how can you tell it's not working? How can you troubleshoot?
    - And how could you do experiments for which there are no kits?
    - Knowing a bit of code lets you troubleshoot, and go beyond kits to create new analyses
- R is one of the most widely-used and powerful programming languages, and it's popular in many areas.

    - R is specifically a staistical language designed for statistical analysis
        - It's got a strong focus on data representation, as you'll see shortly
        - Particularly good for the generation of publication-quality graphs and figures.
    - It is a *general programming language*, though. Most of the concepts you will learn apply to Python and other programming languages.
        - Once you understand programming concepts in one language, it's easier to move across to other languages.

- However, R is not the easiest-to-learn programming language ever created.
    - Please don't get too discouraged! Some bits and concepts *are* just a bit more difficult than others
    - Even with the modest amount of R we will cover today and tomorrow, you can start using some sophisticated R software packages, have a general sense of how to interpret an R script, and be able to get your own data into a workable form for more sophisticated exploration.
- Get through these lessons, and you will be on your way to being an accomplished R user!
- The other big "secret" of programming is that you can only learn so much by reading about it. You have to do it - and get things wrong - to learn.
    - Do the exercises in class, re-do them on your own, and then work on your own problems, and you'll be expert in no time!
- **Let's get started.**

What is R?

- R is a **PROGRAMMING LANGUAGE**, and the **SOFTWARE** that runs programs written in that language.
- R is **AVAILABLE ON ALL MAJOR OPERATING SYSTEMS**
- The way R is used when speaking can sometimes be confusing - **IF AT ANY POINT I AM UNCLEAR, PLEASE ASK!**
- **WHY DO WE USE AND TEACH R?**
    - R is **FREE AS IN "FREE SAMPLES AT THE SUPERMARKET"**, and very **WIDELY-USED** across a range of disciplines.
    - There are **MANY USEFUL PACKAGES** or data analysis and statistics, **WRITTEN By EXPERTS** at the cutting edge of their fields
    - It has excellent **GRAPHICS CAPABILITIES**
    - There is an international, friendly **COMMUNITY** across a range of disciplines, so it's easy to find local and online support

---

What is RStudio?

- RStudio is an **INTEGRATED DEVELOPMENT ENVIRONMENT** - which is to say it's a very powerful tool for writing, using and running R, and programs written in the R language.
    - It's available on **ALL MAJOR OPERATING SYSTEMS**
    - It's available as a webserver, too.
- On the left is a screenshot **TAKEN WHILE I WAS WRITING THIS PRESENTATION IN RSTUDIO** on a Mac
- On the right is the Windows version, with an **EXAMPLE ANALYSIS AND VISUALISATION**
- You can use it to **INTERACT WITH R DIRECTLY TO EXPERIMENT WITH DATA**
- It has a **CODE/SCRIPT EDITOR FOR WRITING PROGRAMS**
- It has tools for **VISUALISING DATA**
- It has built-in **GIT INTEGRATION FOR MANAGING YOUR PROJECTS**

RStudio overview - INTERACTIVE DEMO

- **START RStudio** (click icon/go into start menu and select RStudio/etc.)
    - **CHECK EVERYONE CAN START RSTUDIO**

 Red sticky for a question or issue  Green sticky if complete

- **REMIND PEOPLE THEY CAN USE RED/GREEN STICKIES AT ANY TIME**
- You should see **THREE PANELS**
    - Interactive R console: **you can type here and get instant feedback**
    - Environment/History window
    - Files/Plots/Packages/Help/Viewer: **interacting with files on the computer, and viewing help and some output**
- **REMEMBER THE WINDOWS ARE MOBILE AND PEOPLE COULD HAVE THEM IN ANY CONFIGURATION - THE EXACT ARRANGEMENT IS UNIMPORTANT**
- We're going to use R in the interactive console to get used to some of the features of the language, and RStudio. **DEMO CODE: ASK PEOPLE TO TYPE ALONG**
    - **THE RIGHT ANGLED BRACKET IS A PROMPT: R EXPECTS INPUT**
    - **TYPE THE CALCULATION, THEN PRESS RETURN**

```
> 1 + 100
[1] 101
> 30 / 3
[1] 10
```

- **RESULT IS INDICATED WITH A NUMBER** `[1]` this indicates the line with output in it
- If you type an **INCOMPLETE COMMAND**, R will wait for you to complete it
    - **DEMO CODE**

```
> 1 +
+
```

- The **PROMPT CHANGES TO** `+` **WHEN** `R` **EXPECTS MORE INPUT**
- You can either complete the line, or use `Esc` (`Ctrl-C`) to exit

```
> 1 +
+ 6
[1] 7
> 1 +
+

>
```

- **ARROW KEYS RECOVER OLD COMMANDS**
- **THE** `HISTORY` **TAB SHOWS ALL COMMANDS USED**
- R will report in **SCIENTIFIC NOTATION**
    - **CHECK THAT EVERYONE KNOWS WHAT SCIENTIFIC NOTATION IS**

 Red sticky for a question or issue  Green sticky if complete

```
> 2 / 1000
[1] 0.002
> 2 / 10000
[1] 2e-04
> 5e3
[1] 5000
```

- R has many **STANDARD MATHEMATICAL FUNCTIONS**
- **FUNCTION SYNTAX**
    - type the function name
    - open parentheses
    - type input value
    - close parentheses
    - press return
    - **DEMO CODE**

```
> sin(1)
[1] 0.841471
> log(1)
[1] 0
```

- We can do **COMPARISONS** in R
  - Comparisons return TRUE or FALSE. **DEMO CODE**
  - **NOTE:** when comparing numbers, it's better to use `all.equal()` (*machine numeric tolerance*) **ASK IF THERE'S ANYONE FROM MATHS/PHYSICS**

```
> 1 == 1
[1] TRUE
> 1 != 2
[1] TRUE
> 1 < 2
[1] TRUE
```

- **THE ORDER/CONSTRUCTION OF MATHEMATICAL OPERATIONS CAN MATTER**
  - Write somewhere if possible: $a = \log(0.01^{200})$, $b = 200 \times \log(0.01)$
  - These two mathematical expressions are exactly equal: $a = b$
  - But computers are not mathematicians, they're machines. Numbers are susceptible to this *rounding error*, so what happens is this:

```
> log(0.01^200)
[1] -Inf
> 200 * log(0.01)
[1] -921.034
```

- **COMPUTERS DO WHAT YOU TELL THEM, NOT NECESSARILY WHAT YOU WANT**

[SHOW SLIDES]

Variables

- **VARIABLES** are critical to programming in general, and also to working in R
- To do interesting and useful things, we need a way to label and refer to a piece of data
  - **We use *variables* for this**
- Variables are like **NAMED BOXES**
  - Like a box, they **HOLD THINGS**
  - When we make reference to a box's name, we **MEAN THE THING THE BOX CONTAINS**
- Here, the box is called Name, and it contains the word Samia

- When we refer to the box, we call it `Name`, and we might ask questions like:
    - "What is the length of `Name`?", meaning "What is the length of the word in the box called `Name`?" (answer: 5)
- Like boxes, a variable **CAN BE EMPTY**
- We can also take the contents out and replace them with something else, but **KEEP THE NAME**

[INTERACTIVE DEMO IN RSTUDIO]

- This is a very important concept, so we're going to go through some practical examples, to reinforce it
    - In `R`, variables are assigned with the **ASSIGNMENT OPERATOR** `<-` (called "left arrow")
    - We will assign the value `1/40` to the variable `x`
    - **DEMO CODE**

```
> x <- 1 / 40
```

- At first, nothing seems to happen, but we can see that the variable `x` now exists, and contains the value `0.025` - a **DECIMAL APPROXMATION** of the fraction `1/40`

```
> x
[1] 0.025
```

- We can also print the output by putting the expression in parentheses

```
> (x <- 1/40)
[1] 0.025
```

- **CLICK ON THE ENVIRONMENT WINDOW**
    - You should see that `x` is now defined there
    - The *Environment* window in `RStudio` tells you the name and content of every variable currently active in your `R` session.
- This **VARIABLE CAN BE USED ANYWHERE THAT EXPECTS A VALUE**

```
> x + x
[1] 0.05
> 2 * x
[1] 0.05
> x ^ 2
[1] 0.000625
```

- such as an argument to a function

```
> log(x)
[1] -3.688879
> sin(x)
[1] 0.0249974
```

- We can **REASSIGN VALUES TO VARIABLES**
  - **MONITOR THE VALUE OF `x` IN THE ENVIRONMENT WINDOW**
  - We can also assign a variable to itself, to modify the variable

```
> x <- 100
> x <- x + 5
> x
[1] 105
```

- We can assign **ANY KIND OF VALUE** to a variable
  - Including **STRINGS** - i.e. sets of characters

```
> name <- "Samia"
> name
[1] "Samia"
```

[BACK TO SLIDES]

Naming Variables

- **VARIABLE NAMES ARE DOCUMENTATION**
  - they should describe what the data represents
    - I expect you can tell, just by reading, what all these variables represent
  - **This makes the code more readable** - you can track the operations on a specific bit of data
  - ("anonymous" variables are also helpful, e.g. `x` or `i`)
- There are some rules and guidelines for allowed and effective variable names
  - The variable names should be explicit, but not too long
  - In `R` names **cannot** start with a digit
  - `R` is also case sensitive, so `Weight` with a capital letter is not the same name as `weight` with a lower-case letter
  - Problems will arise if you call any variables by the name of an existing `R` function
  - You should use consistent styling to handle multiple words

---

Functions

- You've already used some **functions** (`log()`, `sin()`, etc.). These are like "canned scripts"
- Functions have **THREE MAIN PURPOSES**
    - They **ENCAPSULATE AND AUTOMATE COMPLEX TASKS** - you don't need to worry about how a sine is calculated, just that you give the function a value, and it tells you what the sine is.
    - They **MAKE CODE MORE READABLE** - by dividing code into logical operations, represented by short names that describe an action, the code is easier to read
    - They also **MAKE CODE MORE REUSABLE** - you don't need to write the routine for finding a square root every time you want one, you just need to call the `sqrt()` function
- Functions usually **TAKE ARGUMENTS** (input), e.g. `sqrt(4)` - the `4` is an argument
- Functions often **RETURN** values (output), e.g. `sqrt(4)` **returns** the value `2`
- **ALL THE FUNCTIONS YOU'VE SEEN SO FAR ARE BUILT-IN**, the so-called `base` functions
- **YOU CAN WRITE YOUR OWN FUNCTIONS**
- **OTHER FUNCTIONS FOR SPECIFIC TASKS CAN BE BROUGHT IN, THROUGH `libraries`**

Getting Help in R: INTERACTIVE DEMO

[DEMO IN CONSOLE]

```
> args(lm)
function (formula, data, subset, weights, na.action, method = "qr",
    model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
    contrasts = NULL, offset, ...)
NULL
> ?sqrt
> help(sqrt)
> ??sqrt
> help.search("sqrt")
> help.search("categorical")
> vignette(two-table)
Error in vignette(two - table) : object 'two' not found
> vignette("two-table")
```

Challenge 01

Link: https://strathsci.qualtrics.com/jfe/form/SV_eG17F5atskDqbC6

Solution:

`mass <- 47.5` This will give a value of 47.5 for the variable mass

`age <- 122` This will give a value of 122 for the variable age

`mass <- mass * 2.3` This will multiply the existing value of 47.5 by 2.3 to give a new value of 109.25 to the variable mass.

`age <- age - 20` This will subtract 20 from the existing value of 122 to give a new value of 102 to the variable age.

 Red sticky for a question or issue  Green sticky if complete

---

3. PROJECT MANAGEMENT IN R

How Projects Tend To Grow

- I'm sure you all recognise this situation
- Research tends to be incremental
    - Projects start out as random notes here, a bit of code there, some data over there
    - and eventually a manuscript is written
- There are many reasons why we should **ALWAYS** avoid working in a random manner, and naming files like they do in the cartoon.
    - Basically, good practice ensures reproducibility and makes everyone's lives easier.

Good Practice

- There is **NO SINGLE GOOD WAY TO ORGANISE A PROJECT**
    - It's important to find something that **WORKS FOR YOU**
    - But it's **ALSO IMPORTANT THAT IT WORKS FOR COLLABORATORS**
    - Some **PRINCIPLES MAKE THINGS EASIER FOR EVERYONE**
- Using a **SINGLE WORKING DIRECTORY PER PROJECT/ANALYSIS**
    - **NAME IT AFTER THE PROJECT**
    - **EASY TO PACKAGE UP** the whole directory and move it around, or share it - **OR PUBLISH ALONGSIDE YOUR PAPER**
    - **NO NEED TO HUNT AROUND THE WHOLE DISK** to find relevant or important files.
    - You can use **RELATIVE PATHS** that will always work, so long as you work within the project directory
- Treat your **RAW DATA AS READ-ONLY**
    - Establishes **PROVENANCE** and **ENABLES REANALYSIS**
    - Keep raw data in a **SEPARATE SUBFOLDER**
- **CLEAN THE DATA PROGRAMMATICALLY** (part of the analysis chain)

    - Most of the time, your data will be "dirty" - it won't be in the form necessary for your analysis.
    - It is often said that 80% of data analysis is spent on the process of cleaning and preparing the data
    - Remove/fill in null values, etc. - whatever is appropriate
    - **KEEP CLEANED DATA SEPARATE FROM RAW** - like food hygiene

- **GENERATED OUTPUT IS DISPOSABLE**

    - This means **ANYTHING GENERATED AUTOMATICALLY BY YOUR CODE/ANALYSIS**
    - If a file can be generated by scripts/code in your project, no need to put it under version control
    - keep in its own subdirectory

Example Directory Structure

- This is **ONE OF MANY WAYS TO STRUCTURE YOUR WORKING DIRECTORY**

    - It's a good starting point, but something else might be more appropriate for your own work

- `WORKING DIR/` is the *root* directory of the project.

    - Everything related to the project (subdirectories of data, scripts and figures; `git` files; configuration files; notes to yourself; whatever)
    - Name this after your project

- `data/` is a subdirectory for storing data

    - raw data only, or raw and intermediate data? **YOUR DECISION**
    - store data and metadata (data *about* the data) together
    - `data/raw`, `data/intermediate` - **USE SUBFOLDERS WHEN SENSIBLE**

- `data_output/` could be a place to write the analysis output (`.csv` files etc.)

- `documents/` is a place where notes, drafts, supporting material, and explanatory text could be stored

- `fig_output/` could be a place to write graphical output of the analysis (keep separate from tables)

- `scripts` might be where you would choose to keep the executable code that automates your analysis

- The important thing is that the structure is **AS SELF-EXPLANATORY AS POSSIBLE**

- General rules to keep in mind

    - **THE PROJECT DETERMINES THE STRUCTURE**: you might have more flexibility with small investigative projects than with safety-critical code
    - **STRUCTURE IS A MEANS TO AN END**: you can bend and break guidelines when it enables robustness, readability, and maintenance
    - **SOME TOOLS CAN HELP**: Linters can make sure your code is typo-free and has no syntax errors; cookie-cutter templating can be useful
    - **ADD A README FILE**

Project Management in `RStudio`

- `RStudio` **TRIES TO BE HELPFUL** and provides the 'Project' concept
    - Keeps **ALL PROJECT FILES IN A SINGLE DIRECTORY**
    - **INTEGRATES WITH** `GIT`
    - Enables switching between projects within `RStudio`
    - Keeps project histories
- We're going to create a **NEW PROJECT** in `RStudio`
- **INTERACTIVE DEMO**
- **CREATE PROJECT**
- Click `File` -> `New Project`
    - Options for how we want to create a project: brand new in a new working directory; turn an existing directory into a project; or checkout a project from `GitHub` or some other repository
- Click `New Directory`
    - Options for various things we can do in `RStudio`. Here we want `New Project`
- Click `New Project`
    - We are asked for a directory name. **ENTER** `ibioic-r-lesson`
    - We are asked for a parent directory. **PUT YOURS ON THE DESKTOP; STUDENTS CAN CHOOSE ANYWHERE SENSIBLE**
- Click `Create Project`
- **YOU SHOULD SEE AN EMPTY-ISH** `RSTUDIO` **WINDOW**
- **INSPECT PROJECT ENVIRONMENT**
- First, **NOTE THE WINDOWS**: editor; environment; files
- **EDITOR** is empty
- **ENVIRONMENT** is empty
- **FILES** shows
    - **CURRENT WORKING DIRECTORY** (see breadcrumb trail)
        - **CHANGE CWD IF NECESSARY**
        - **SHOW TERMINAL**
    - **ONE FILE**: `*.Rproj` - information about your project
- **CREATE DIRECTORIES IN PROJECT**
- **Create directoris called** `scripts` **and** `data`

    - Click on `New Folder`
    - Enter directory name (`scripts`)
    - Note that the directory now exists in the `Files` tab
    - Do the same for `data/`

- **NOTE THAT WE WILL NOW POPULATE THE DIRECTORY**