Introduction to R

Slides inherited from Dr. Georgina Menzies, Dr. Nicos Angelopoulos & Matthew Bracher-Smith

Gabriel Mateus Bernardo Harrington

bernardo-harringtong@cardiff.ac.uk

Cardiff University

UK Dementia Research Institue

Todays Aims

- Basics of R
- Importing and Exporting Data
- Working with R packages
- Basic data alterations

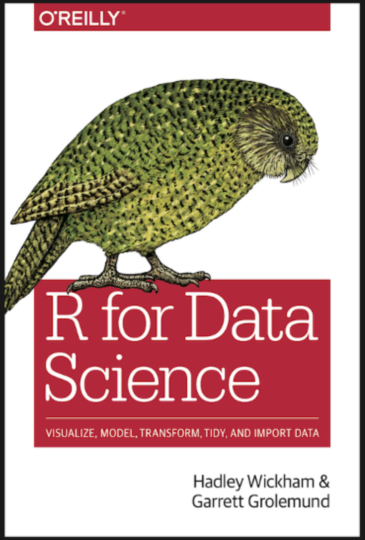
Learning Objectives

- Get familiar with RStudio
- Understand R syntax
- Learn basic calculations
- Explore data types and structures
- Learn data import/export in base R
- Find help and install packages

Recommended Textbook

This course will follow **R for Data Science** (2nd Edition):

- Link to R4DS book
- Major updates: Swap to **Quarto** from R markdown
- Emphasizes data visualization and `ggplot2`
- Excludes modeling (see Tidy Models book for that)



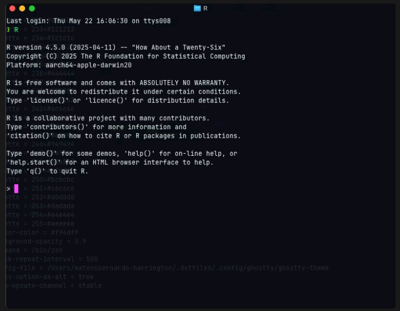
Install R and RStudio

1. **R**: Current version (4.5.1 “Great Square Root”)
   - Download for Windows
   - MacOS (use ARM64 for M1 Macs)
   - Ubuntu Linux: `sudo apt install r-base`
2. **RStudio**: interactive development environment (IDE) for R
   - Write code, analyze data, create plots
   - Download RStudio

You are welcome to use Jupyter Notebooks/lab as well, but note that the course will assume Rstudio and it’s best to familiarise yourself with this first.

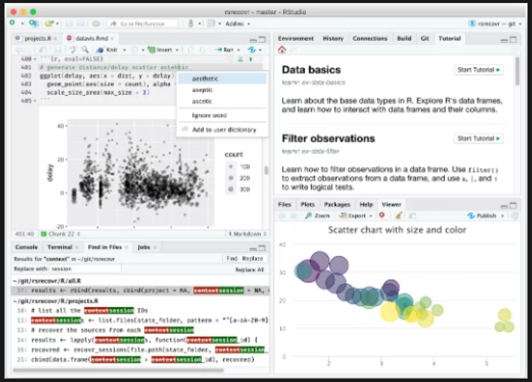
Jupyter vs RStudio

- **R Console**
- Simplest way to interact with R
- No additional features

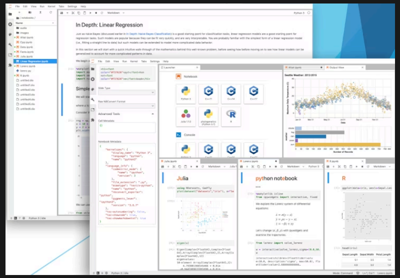


```
Last login: Thu May 22 16:06:30 on ttys008
❯ R

R version 4.5.0 (2025-04-11) -- "How About a Twenty-Six"
Copyright (C) 2025 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20

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

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

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

> 
```

- **RStudio**
- Main IDE we will use
- Includes console, scripts and much more



- **JupyterLab**
- Can use any of Julia, Python or R
- Happy for you to use/try but not the focus of this course



Basic Math in R

```r
2 + 2
```

```
[1] 4
```

```r
5 - 3 + 2 * 4^2
```

```
[1] 34
```

```r
11465 * 2358971436
```

```
[1] 27045607513740
```

```r
options(scipen = 999)
11465 * 2358971436
```

- What is the difference between these last two runs?
- What do you think `options(scipen = 999)` is doing to the code?

options(Scipen)

‘scipen’: integer. A penalty to be applied when deciding to print numeric values in fixed or exponential notation. Positive values bias towards fixed and negative towards scientific notation: fixed notation will be preferred unless it is more than ‘scipen’ digits wider.

Common Math Operators in R

Action	Symbol
Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	^
Modulo	%%

- You can info on math expressions here

Maths test!

Solve these problems in your R console:

- What is 2 to the power of 16?
- What is the square root of 92?
- If I have 7 apples and 16 kids - how much of an apple do they each get?

The Prompt

The prompt is represented by this symbol >.
This means that R is waiting for your input.
This is how you know that nothing is currently running.

If your code isn't correctly formatted, R will "hang", and you will see this symbol instead +.
You will need to kill the line to try again, and you can do this using ESC.

If this prompt is missing you know that R is running your code, depending upon the complexity of the input code this might take some time.

Try running this code in R, what happens?

```r
5 - 3 + 2 * 4 ^
```

What do you think is wrong?

Commenting in R

- R ignores everything after a #
- You should be carefully and thoroughly commenting every bit of your code so that other people (**including future you**) understand what it does
- It can also come after some code:

```r
"Hello_World" # a word/character
```

```
[1] "Hello_World"
```

- 4 hashtags will also tell RStudio to allow code folding at that line, very handy for long scripts!

Getting help in R

R is great at offering extra help.

If you know the function name you can type `?function_name` or `help(function_name)`.
If you aren't quite sure of the function name try `??function_name`!

Try this out and see what you get.

```r
?mean
??mean
help(mean)
```

Searching online and LLMs

- Back in the not so distance past, Stackoverflow was the place to find help with codey matters…
- Now most people will use Large Language Models (LLMs)
  - A word of caution!
  - They can be handy tools, but try to be very intensional about how you use them whilst learning!
  - You'll be using R throughout this course, so it's important you actually built confidence using R, not just pass the assessments
  - If you use them, make sure you understand the output they give you (try asking it to explain parts you don't understand and comment any code it writes well)

Variables

- We want to be able to save numbers, vectors, objects, data frames etc. so that we can call on them again later
- Variable names cannot contain special characters (except `_` & `.`), and they cannot start with a number.

`<-` is used to assign values to a variable
i.e. less than followed by a dash, I like to think of it as you are putting something into your variable name

Variable Assignment

```r
x <- 5
y <- x + 2
print(y)
```

```
[1] 7
```

- Avoid using `=` for assignment to avoid potential scoping issues.

Code formatting matters!

- Your code should be readable!
- Use snake_case for variable names not CamelCase

```r
# Good
day_one <- 1
day_1 <- 2

# Bad
DayOne <- 1
dayone <- 2
```

- Use spacing

```r
x<--4+5/6*4
x <- -4 + 5 / 6 * 4
```

- See the tidyverse style guide for more info, and the styler package for automatic formatting

Data Types

- There are lots of data types in R
- Some function will only work with some data types, and some will give different types of outputs depending on the input data type, so be careful!

Type	Example
Character	‘words’
Numeric	1, 23.25
Integer	2L
Logical	TRUE, FALSE
Complex	1 + 4i

```r
x <- 3
typeof(x)
```

```
[1] "double"
```

Tables, Data Frames, Lists, Vectors

The types of data that can be stored and used in R are collectively called **objects**

Vectors in R

Vectors are multiple objects of the same class in one object.

We use the ‘c()’ function to join them all together.

Make your own Vector like the one below.

```r
apple <- c('red', 'green', 'yellow')
print(apple)
```

```
[1] "red"    "green"  "yellow"
```

```r
# Check the type
print(typeof(apple))
```

```
[1] "character"
```

Have a go!

Try the following:

- Make two vectors of equal length, containing only numbers, save them to new variables
- Add the two vectors together using their variable names, what happened?
- What happens if you type `name_of_your_vector[2]`?

Indexing Vectors

```r
# If we want a specific element
vec <- c("a", "b", "c", "d", "e", "f")
vec[2]
```

```
[1] "b"
```

```r
# If we want a range of elements
vec[3:6]
```

```
[1] "c" "d" "e" "f"
```

```r
# If we want multiple specific elements
vec[c(2, 5)]
```

```
[1] "b" "e"
```

Vectors only contain one type of data

- Vectors can only contain a single type or data, so you can’t have both characters and numerics together in one vector
- What do you think happens if you try?

```r
# What will the type of confused be?
confused <- c(1, 2, "3")
```

```r
print(confused)
```

```
[1] "1" "2" "3"
```

```r
typeof(confused)
```

```
[1] "character"
```

- There is an order of type conversion that R uses, with character being the broadest type
- It’s common for columns you expect to be numeric to end up being characters due to a rouge value, be careful!

Operators

- Relational operators

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

- Logical operators

Operator	Description
!	NOT - flips TRUE/FALSE
&	AND - TRUE if both condittions for each element
&&	AND - Same as above but for **first** element
|	OR - TRUE if at least one condition is TRUE for all elements
||	OR - Same as above but for **first** element

Operator examples

```r
num1 <- c(TRUE, FALSE, 0, 23)
num2 <- c(FALSE, FALSE, TRUE, TRUE)

# Performs AND operation on each element in both num1, num2
num1 & num2
```

```
[1] FALSE FALSE FALSE  TRUE
```

```r
# Performs OR operation on each element in both num1, num2
num1 | num2
```

```
[1]  TRUE FALSE  TRUE  TRUE
```

```r
# This will convert all the num1 TRUE values to FALSE, and FALSE values to TRUE
!num1
```

```
[1] FALSE  TRUE  TRUE FALSE
```

```r
# From num2 Vector - This will convert all the TRUE values to FALSE, and FALSE to TRUE
!num2
```

```
[1]  TRUE  TRUE FALSE FALSE
```

Logicals are secretly numerics

- Try adding two logicals, what happens?

```r
# What is the value of secret?
secret <- TRUE + TRUE - FALSE
```

```r
print(secret)
```

```
[1] 2
```

- FALSE is actually considered to be 0 and TRUE is 1
  - Note that TRUE is technically any non-0 value, but normally this doesn't matter

```r
madness <- c(TRUE, FALSE, 0, 23, -5)
typeof(madness)
```

```
[1] "double"
```

```r
print(madness)
```

```
[1]  1  0  0 23 -5
```

```r
as.logical(madness)
```

```
[1]  TRUE FALSE FALSE  TRUE  TRUE
```

Why this is useful

This is handy because it lets us do math with logicals!

```r
# Get a random sequence of 10 values between 1 and 100
numbers <- sample(1:100, size = 10)
print(numbers)
```

```
 [1] 31 46 87 67 39 33 50 69 89 76
```

```r
# How many values are below 50?
numbers < 50
```

```
 [1]  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
```

- We can just add up the logicals

```r
sum(numbers < 50)
```

```
[1] 4
```

Lists

A list is an R-object which can contain many different types of elements inside it like vectors, functions and even another list inside it.

```r
list1 <- list(c(2, 5, 3), 21.3, sin, list(1:5, c("a", "b")))
print(list1)
```

```
[[1]]
[1] 2 5 3

[[2]]
[1] 21.3

[[3]]
function (x)  .Primitive("sin")

[[4]]
[[4]][[1]]
[1] 1 2 3 4 5

[[4]][[2]]
[1] "a" "b"
```

Matrices

A matrix is a two-dimensional rectangular data set. It can be created using a vector input to the matrix function.

```r
M <- matrix(c('a', 'b', 'c', 'd', 'e', 'f'), nrow = 2, ncol = 3, byrow = TRUE)
print(M)
```

```
     [,1] [,2] [,3]
[1,] "a"  "b"  "c" 
[2,] "d"  "e"  "f" 
```

- Change byrow = FALSE to see what happens.

Data Frames

- Data frames are tabular objects with columns of different types.

Now you’ve learnt a little more about getting help in R. Can you find out how to make a data frame?

Data frames continued

```r
df <- data.frame("letters" = c("a", "b", "c"), "numbers" = c(1, 2, 3))
print(df)
```

```
  letters numbers
1       a       1
2       b       2
3       c       3
```

- Use `$` to reference entire columns.

```r
df$numbers
```

```
[1] 1 2 3
```

```r
# This is handy for applying functions to columns
sum(df$numbers)
```

```
[1] 6
```

Subsetting Data

- You can index with ‘[]’

```r
# The first value is the row and the second is the column
# So the following is the thirds row, first column
df[3, 1]
```

```
[1] "c"
```

- Use indexing and logical conditions to subset data:

```r
subset <- df[df$numbers == 2, ]
print(subset)
```

```
  letters numbers
2       b       2
```

Break Time

R Functions

- You will learn how to properly write and use functions later
- These are ways of saving time writing code you plan to use over and over again
- R has many built-in functions but there are millions more other people have written as well.

```r
sum_of_squares <- function(x, y) {
  x^2 + y^2
}

sum_of_squares(3, 7)
```

```
[1] 58
```

Installing Packages

- R is open source and easily expandable. Therefore a lot of people have contributed packages to R over the years. These are functions people have written to perform a wide variety of tasks.
- You can find a very long list of packages here.
- There is a biology specific set of packages for R called BioConductor, and a data science one called the tidyverse (and this we will be coming back to).
- You can also write your own package and submit it to the R community. We will come back to looking at packages and the writing of packages later in the course, but if you are interested there is a book about it! You can find it online here.
- Hilary Parker's posts on personal R packages and how to create one

Installing Packages continued

There are a couple of ways to install packages into R, but the easiest way is just to install them through the console and load them in like this:

- Install the package.
  - This downloads the required code from the web repository (by default this should be “https://cran.rstudio.com/”). Packages may require other packages to work, therefore more than one package may downloaded.

```r
install.packages('packagename')
```

- Load the package into R.
  - Once the packages is downloaded, it then needs to be loaded into the R environment for use. Downloading the package does not make the commands available, it must be loaded into R first.

```r
library(packagename)
```

Try installing a package!

- Try and install the `tidyverse` and load it

- Install packages from CRAN:

```r
install.packages('tidyverse')
```

- Load a package:

```r
library(tidyverse)
```

Working Directories

- Files you save have to go somewhere on your computer
- R needs to know where is should save stuff
- This can be done in two main ways:
  - Absolute paths (full system path)
  - Relative paths (path relative to other directories/files)
- Examples:
  - `C:/Users/matt/Dropbox/Teaching/Lesson 1 – Into to R/` - typical absolute path on Windows
  - `03_data/my_data.csv` - relative path from project root

Working Directories - continued

- Use getwd() and setwd() to manage your working directory.

```r
setwd("C:/path/to/directory")
```

- Strongly suggest avoiding changing working directory in scripts, try to keep to relative file paths, as the absolute path is unique to your environment
  - If you do have absolute paths, the code won't run on another computer and the code will break!

- Check out the here package to help manage relative file paths
  - It helps figure out the project root based on the presence of a `.Rproj` file and/or a `.git` directory

```r
here::here()
```

```
[1] "/Users/mateusbernardo-harrington/UK Dementia Research Institute Dropbox/Gabriel Bernardo
Harrington/backup/teaching/bioinformatic_masters_r_lectures"
```

Importing Data

- Data can be in many formats (comma separated file (.csv), Excel (.xlsx), tab delimited (.tsv or .txt))
- The function you use to read in data will depend on the format of the file

```r
# base R csv - note using here for relative paths!
data <- read.csv(here::here("03_data/data.csv"))
# tab and space delimited
data <- read.table("data.tsv", sep = "\t")
data <- read.table("data.txt", sep = " ")

# Excel
library(readxl) # don't forget to install the package if needed!
data <- read_excel("data.xlsx", sheet = 1)
```

Importing Data - continued

- Other considerations:
    - Does the data have column headers
    - Are there leading lines to be skipped?
    - Is there a non-standard identifier for missing data (NA, -9, -, etc.)
    - What character is used as a decimal point?
- How can we check the arguments we could add to these functions?
- Note that there are other packages for reading in data that can be more performant, and have different default behaviour (e.g. `readr` or `data.table`)
- This can be particularly important to consider when reading in larger files

Exporting Data

- You’ll likely want to save the data you’ve been working with at some point
- Similarly to importing data, you’ll use different functions depending on the format you want to save to

```r
df <- mtcars # build in dataframe in R

# Save to csv
write.csv(
  df,
  here::here("03_data/mydata.csv"),
  row.names = FALSE,
  quote = FALSE
)

# Save to tsv
write.table(df, here::here("03_data/mydata.tsv"), sep = "\t")
```

- Use row.names = FALSE and quote = FALSE for better formatting.

Additional Resources

- Posit Cheatsheets
  - Cheatsheets for various packages and RStudio itself (by developers of RStudio)
- Free online book R for Data Science
- These slides and the workshop can be found on the website here: