PY1617: Foundations of R for Statistics and Data Science

Component Notebook

Dr Abigail Page & Dr James Winters

Contents

1 Introduction to R: Week 1

> Welcome to the R component of **PY1617 Employability: Foundations of R for Statistics and Data Science**

This notebook brings together all material for the R component of the module, spanning ten weeks of teaching.

In this first week, the focus is on **orientation**: understanding what R is, why we are using it, how it fits into data science and statistics and getting everything set up correctly on your computer.

1.1 Overview of the R Component

This component runs across **ten teaching weeks**, starting from the very basics of using R and progressing to writing functions and simulating data by the end of the term.

Teaching is shared across the module:

1) **Weeks 1–5** are taught by Dr Abigail Page and focus on foundational R skills, data handling, and visualisation
2) **Weeks 7–10** are taught by Dr James Winters and focus on programming concepts and simulation

By the end of the component, you will be able to use R confidently as a tool for working with data in academic, research and workplace contexts.

1.2 Aims of the Component

The R component aims to develop **foundational skills in R** that underpin data analysis, statistics and data science.

Specifically, the component aims to: 1) Build confidence in using R and RStudio 2) Develop good habits for writing clear, reproducible code 3) Introduce core data workflows used in statistics and research 4) Prepare you for later statistics modules in Years 1 and 2 5) Develop digital and data literacy skills valued in the workplace

---

1.3 Learning Objectives

By the end of this component, you should be able to:

1) Use **R and RStudio** confidently, including scripts, projects and packages

2) Understand and work with **data types, objects and data frames** in R

3) Apply a **tidy data workflow** to clean, transform and reshape data

4) Create and label **new and recoded variables**

5) **Summarise and visualise data** effectively using ggplot2

6) Use basic **programming concepts** (loops and functions) to automate tasks

7) Write **clear, reproducible code** suitable for academic and workplace contexts

8) Use these skills to **simulate data**

1.4 What Is Data Science?

Data science is the process of turning data into understanding and insight.

This typically includes:

1) Managing data

2) Cleaning and preparing data

3) Exploring and visualising data

4) Modelling and interpreting results

5) Communicating findings clearly

Data science is best thought of as a **workflow**, rather than a single technique.

1.4.1 The data science workflow

Throughout this module, we will refer to the **data science cycle (Figure 1.1)**, which describes how data are:

1) Imported: bringing data into R

2) Prepared and transformed: cleaning the data so you can do what you want with it.

3) Explored and visualised: explore patterns and trends

4) Modelled: apply statistical models

5) Communicated: report your results clearly in plots and tables

We can think of this as a cycle because you don't do the steps only once, but you loop back and forth.

The first step in any data analysis is importing your data into R. This usually involves loading data from a file into a data frame. Without access to your data within R, you cannot perform any analysis. Once your data is imported, it's important to tidy it. Tidy data has a consistent structure where each column represents a variable and each row represents an observation and each cell represents a value (Figure 1.2). This organization makes it easier to work with the data, allowing you to focus on analysis rather than data cleanup.

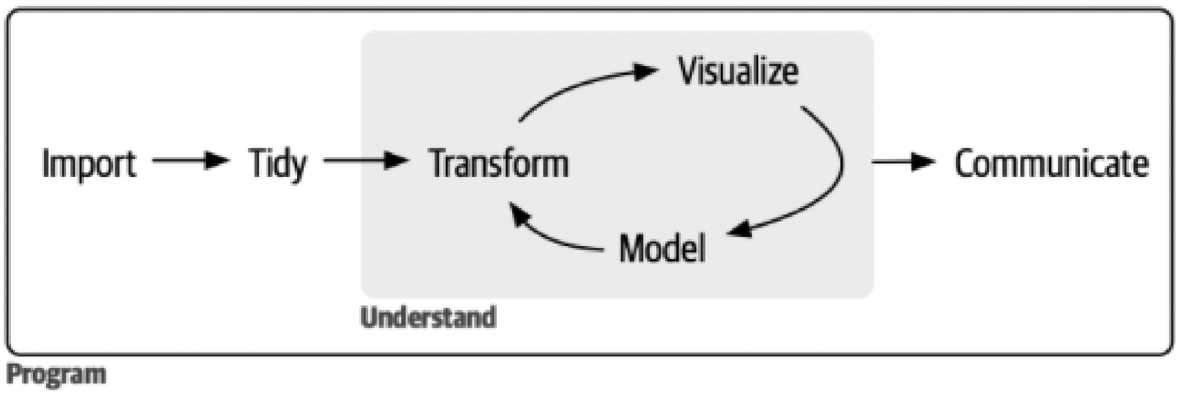

Figure 1: Figure 1.1 Workflow Diagram from R4DS textbook

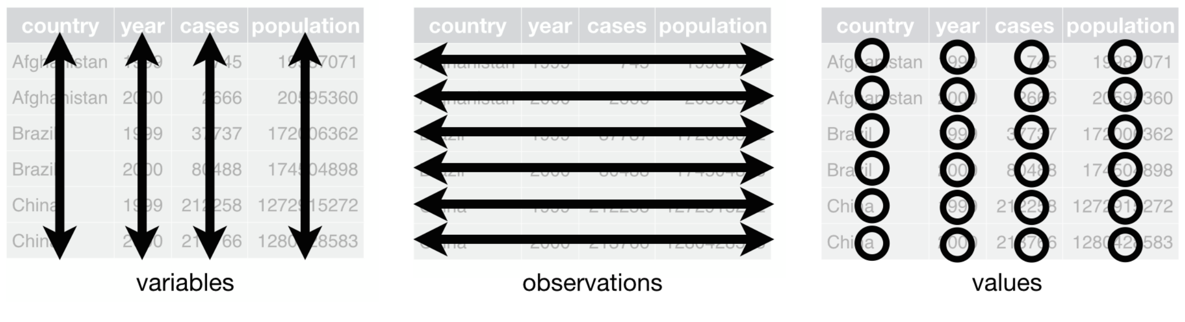

Figure 2: Figure 1.2 Variables, observations and values from R4DS textbook

After tidying, the next step is often transforming your data. Transformations can include filtering for specific observations (e.g., data from a particular participant or year), creating new variables from existing ones (e.g., calculating speed from distance and time), or computing summary statistics (e.g., counts, averages). The process of tidying and transforming data is commonly called data wrangling, because getting data into a usable form can be challenging.

With clean and well-structured data, you can begin generating insights through visualisation and modelling. These approaches complement each other:

- Visualisation is a human-centred activity. Well-designed visualisations can reveal unexpected patterns, suggest new questions or indicate that you may need additional data. While visualisations are invaluable for understanding, they rely on human interpretation and don't scale easily to large datasets.
- Modelling provides a computational approach. Once your questions are clearly defined, models can answer them efficiently and at scale. However, models operate under assumptions and cannot challenge those assumptions on their own. They are powerful, but inherently limited in their ability to surprise.

Finally, communication is essential. The insights you derive from analysis - whether through visualisations, models or summaries — are only useful if you can clearly explain them to others.

Throughout the process, programming supports every step. While you don't need to be a programming expert, stronger programming skills allow you to automate tasks, work more efficiently and tackle new problems more effectively. These tools cover most of what you'll need in a data science project/

This R component focuses primarily on the **foundational stages of this cycle: importing, tidying and visualising data**. These are the skills that everything else depends on. Data modelling and communication will be focused on in other components and modules.

1.5 What Is R?

R is a programming language designed specifically for **data analysis, statistics and graphics**.
It was first developed in 1996 and is now widely used in academia, research and industry.

R is: - A **command-driven** language - Highly flexible and extensible - Widely used for statistical analysis and data visualisation

You may have encountered other statistical software such as SPSS, Stata, or MATLAB. R differs from these in that it is: - Script-based rather than menu-driven (you write things rather than click things) - Highly customisable - Open source and free to use

---

1.6 What Is RStudio?

RStudio is the software environment we will use to work with R.

You can think of: - R as the engine which does the work
- RStudio as the dashboard which makes it easy to do the work

RStudio provides: - A script editor - A console for running code - Tools for viewing data, plots and files - Integrated help and documentation

Once we have installed RStudio we no longer need to directly access R!

1.6.1 Why we use R & RStudio

R & RStudio is widely used because: - Faster & more accurate than hand calculations: Computers handle repetitive statistical tasks easily. Doing a few calculations by hand helps you understand concepts, but mostly it's about intuition. - Better than spreadsheets: Excel and similar tools feel familiar but are limited for serious analysis. R is far more flexible and powerful. - Free & open source: No expensive licenses, no student-version traps—R is professional-level software at no cost. - Highly extensible: Thousands of additional packages are freely available, letting you use cutting-edge methods and advanced techniques as you grow. - Learn programming along the way: Using R teaches programming skills, which are essential for modern research, online experiments, automated data collection, AI, and computational methods. - Widely used in research: Learning R gives you access to standard tools in statistics and psychology, putting you closer to advanced, real-world methods.

Bottom line: R can be challenging at first, but its power, flexibility, and relevance make it the best choice for serious data analysis.

1.7 Challenges of Learning R

Learning R can feel challenging at first, especially if you do not have a programming background.

Common difficulties include: - A steep initial learning curve - Small errors in code causing unexpected problems (e.g. a missing closing bracket or misplaced comma can be frustrating) - Not knowing how to phrase questions when searching for help

This is normal.

> **A useful rule of thumb:**
> Knowing what to search for is a large part of working effectively in R. AI tools make this a lot easier!

Over time, you will build familiarity with common patterns and solutions.

1.8 Installing R and RStudio

Before you can start using R, you need to install **both R and RStudio**.

1.8.1 Installing R

1. Go to: https://cran.r-project.org

2. Select your operating system (Windows, macOS, or Linux)

3. Download and install the **latest version** of R

   Always install R **before** installing RStudio.

1.8.2 Installing RStudio

1. Go to: https://www.rstudio.com/products/rstudio/download/

2. Download **RStudio Desktop (free version)**

3. Install the latest version for your operating system

Once installed, you will **open RStudio**, not R itself.

Opening RStudio automatically starts R in the background.

Finally You will need to download Rtools from Cran: https://cran.r-project.org/bin/windows/Rtools/. Make sure this is installed before moving on.

1.9 First Steps in RStudio

When you open RStudio for the first time, you will see several **panes** (also called panels). Each has a different purpose to help you work efficiently in R.

- The **console** (where code runs) on the left
- The **environment** (where objects are stored) on the top right
- Panels for plots, files, and help on the bottom right

1.9.1 Console

The **console** is where you **type and run R commands directly**. We will play around with this at the end of the session. The console executes your commands immediately and shows the results.

1.9.2 Envrionment and history

The Environment tab shows all the objects you have loaded into R or created (variables, data frames, lists, etc). The History tab keeps a record of all the commands you have run in the console. You can click on it and see what you have just done, and re-run it.

1.9.3 Files, plots, packages, helps and viewer

This is were you can view lots of useful pieces of information and access your files:

1) Files: Lets you browse files in your project folder. You can import data from here.
2) Plots: Displays graphs and visualisations you create. This pops up automatically when you make a plot.
3) Packages: Displays which R packages are installed and loaded. What you can use to install more packages (we will get to this)
4) Help: Shows documentation for functions you search.
5) Viewer: Used for HTML outputs and interactive visualisations.

These tabs help you navigate your project, visualize results and find help quickly. We will come to each of these sections in turn.

First, there is a final section of RStudio which is only viable when you open up a R Script.

1.9.4 R scripts

An R script is a file where you can write, save and edit multiple lines of R code. Like a word document for R code! Scripts have the file extension .R and are saved in a folder on your computer.

You can either open a blank script: - Go to File (top left piece of paper with a green plus) - New File - R Script

This creates a script where you can write and save your code.

OR you can load a pre-made R script which you or someone else have made to run. You can download the pre-made R Script for Week 1 session here:

Download the R Script

1.10 Basic R Expressions

You can: - Type directly into the **console**, or
- Write code in a **script**, which can be **saved, shared and reproduced**.

> Tip: If you write directly in the console, your work is **not saved** when RStudio closes.

So while we will show you how to write script directly into the console today, in future we will be working with scripts.

1.10.1 Writing in the console

In R, `<-` is the **assignment operator** (not `=`).
- Example: `x <- 500` tells R "x equals 500."
- Whenever you type `x`, R will read it as 500.

Steps: 1) In the console (after the `>`), type:
2) "x <- 500" 3) press enter 4) write x in the next line 5) press enter

it will look like this:

```
x <- 500
x
```

```
## [1] 500
```

> **Question**: when you select "x" what is the output?

> **Activity**: practice with different numbers and letters

In the top right corner of the console you will see a **brush** icon. If you click on it, it clears your console environment. Do this now.

You now have a blank screen so have lost your work. If you go to the history tab now you can see your previous code, and doubling clicking it will bring it back to the Console for you to re-run!

But because you cannot save your work in the console we recommend you always write code in an R script.

1.10.2 Writing in a script

Lines starting with **#** are **comments**. These are notes you leave for yourself or others to help understand what it is you have done

- R **ignores them**, so they are not run.

- They can say anything you like
- They are essential - you will thank yourself later for a well commented script!

Below is what working in a script looks like.

```
 #To run the code below just select the lines and press 'run' above (or control/command + enter).
# You can run line by line, or select it all at once.

x <- 10 # telling R that x now equals 10  (this FYI is an example of a comment on code - the right of t
print (x) # printing this in the console
```

```
## [1] 10
```

```
# so this has done the same as above, but now your work is saved here.

y <- 10 + 20 # telling R that y now equals 10 + 20
print (y)  # printing this in the console - you can run both these lines at once by selecting both.
```

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

```
z <- "hello" # telling R that z now equals "hello"
print(z) # printing this in the console
```

```
## [1] "hello"
```

```
# you can also get R to print by highlighting the text and press enter/run (called auto-printing) which

a <- "hello"
b<- "world"
a
```

```
## [1] "hello"
```

```
b   # run all four lines at once
```

```
## [1] "world"
```

1.11 Working with R: Key things to know

Some final points about R's behaviour, errors, spacing and "unfinished commands"

1.11.1 R is literal

```
- R does exactly what you tell it. It has no intuition, no autocorrect and no ability to guess your inte
- Typos matter. For example 10 = 20 gives you an error
```

```r
10 - 20   # produces -10, but may not be what you meant
```

```
## [1] -10
```

```
- Even simple mistakes can produce wrong answers without errors. Always double-check what you type.
```

1.11.2 R ignores some spacing

```
- extra spacing usually doesn't matter
```

```r
10 + 20
```

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

```r
10+20    # both work fine
```

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

- but don't split words or function names (e.g. you can't say citat ion())
- consistent formatting is good practice

1.11.3 R can comestimes detect incomplete commands

- If R thinks a command isn't finished, it waits for more input
- This also works works for long commands broken over multiple lines
- To cancel a command mid-typing, press escape to return to the prompt.

R is a powerful tool, but it's precise and literal. Be careful with typos, spacing, and parentheses, and remember that errors are part of learning. With practice, reading and understanding R's responses becomes easier.

2 Workflow in R: Week 2

2.1 Files, folders, and projects

2.2 Working directories

2.3 R scripts and packages

3 Data in R: Week 3

3.1 Data types and objects

3.2 Vectors and data frames

3.3 Functions in R

4 Data Manipulation in R: Week 4

4.1 The tidyverse

4.2 Recoding and transforming variables

4.3 Missing data (NAs)

4.4 Wide and long data

5 Data Visualisation in R: Week 5

5.1 Summarising data

5.2 Introduction to ggplot2

5.3 Interpreting plots

6 Conditional Programming: Week 6

6.1 Logical values

6.2 if and else statements

7 Pipes and Loops: Week 7

7.1 Pipes (%>%)

7.2 for loops

7.3 Repeating operations

8 Writing Functions from scratch: Week 8

8.1 Why write functions?

8.2 Function structure