Lab 01 – Getting started

ENVX1002 Handbook

Semester 1, 2025

Learning Outcomes

At the end of this practical students should be able to:

- Install and set up R and RStudio on their computer
- Understand the relationship between R and RStudio
- Create and manage R projects effectively
- Write and render basic Quarto documents
- Import data from CSV and Excel files
- Perform basic operations in R

Settling in

At the beginning of the next few weeks we will be doing some short activities before getting into the stats to help you foster a sense of belonging, learn more about your peers, and help better prepare you for your studies. This week we will start with a simple introduction, but before we do this, we would like to acknowledge those who were here before us:

> *We would like to acknowledge and pay respect to the traditional owners of the land on which we meet; the Gadigal people of the Eora Nation. It is upon their ancestral lands that the University of Sydney is built. As we share our own knowledge, teaching, learning and research practices within this university may we also pay respect to the knowledge embedded forever within the Aboriginal Custodianship of Country.*

To learn more about why we do Acknowledgement of Country, and the difference to Welcome to Country, see the following page: Welcome and Acknowledgement.

AnswerGardens

We are all from diverse backgrounds and have followed different paths to get to where we are today. To help you get to know your peers, your demonstrator will lead a class discussion, posting a number of questions on AnswerGarden, where you can then anonymously post your answer to the questions. Links will be provided once your demonstrator has set up the question.

After about 20 minutes of discussion, we can get started on the lab! Welcome to ENVX1002!

Exercise 1: setting up

Installing R and RStudio

Before we begin working with data, we need to set up our statistical computing environment. We'll be using two main pieces of software:

1. **R**: The statistical programming language (the engine)
2. **RStudio**: The integrated development environment (IDE) that makes working with R easier (the interface)

Think of R and RStudio like a car: R is the engine that does all the work, while RStudio is the dashboard and controls that make it easier to drive.

This week's tutorial would have guided you through the installation process, but if you missed it, below are the steps to install R and RStudio on your personal computer.

> ! Important
>
> You must install R **before** installing RStudio, as RStudio needs R to function.

Installing R

1. Go to the CRAN (Comprehensive R Archive Network) website
2. Click on the link for your operating system
3. Follow the installation instructions for your system
   - For Windows: Click "base" then download the latest version
   - For Mac: Choose the appropriate .pkg file for your system (Apple Silicon or Intel)
   - For Linux: Follow the instructions for your distribution

Installing RStudio

1. Visit the Posit download page
2. Scroll down to "RStudio Desktop"
3. Click the download button for your operating system
4. Run the installer and follow the prompts

> Tip
>
> If you're using a University computer, both R and RStudio should already be installed. However, it's important to install them on your personal computer for working outside of class.

Creating your first R project

An R project is like a container that keeps all your work organised and tidy. Think of it as a dedicated workspace for your course where everything stays together and works smoothly. Here's why R projects are especially helpful for beginners:

Good project organisation is crucial for reproducible research. It helps you and others find files easily and ensures your code works consistently.

- **Consistent starting point**: Every time you open your project, you'll be in the right place with all your files readily available
- **No more lost files**: Your data, code, and outputs stay together in one organised location
- **Easier file paths**: You don't need to worry about complex file locations - R projects make it simple to find and use your files
- **Collaboration ready**: When sharing your work, everything stays organised and works on other computers
- **Better workflow**: As you learn more complex analyses, having an organised project structure will save you time and prevent headaches

Let's create a project for this course:

1. Open RStudio
2. Click File → New Project
3. Choose "New Directory"
4. Click "New Project"
5. Enter "ENVX1002" as the directory name
6. Choose a location on your computer (preferably in a cloud-synced folder)
7. Click "Create Project"

To keep things simple but organised, let's create one essential folder:

```
ENVX1002/
├── ENVX1002.Rproj  # This file helps RStudio manage your project
└── data/           # Store your datasets here
```

To create the data folder, any of the following works:

1. In RStudio's Files pane (bottom-right), click "New Folder", then name it "data"
2. In the console, run `dir.create("data")` to create the folder
3. Manually create a folder named "data" in your project directory, using Finder(macOS), File Explorer(Windows), or similar

> **Tip**
>
> As you progress in the course, you can create more folders to organise your work. But for now, keeping it simple will help you focus on learning R without getting overwhelmed by complex folder structures.

Exercise 2: using Quarto

What is Quarto?

Quarto is a modern publishing system that allows you to:

Quarto documents combine code, text, and output in one file, making your analysis reproducible and easy to share.

- Combine text, code, and output in one document
- Create professional reports, presentations, and websites
- Work with multiple programming languages (including R)
- Generate high-quality output in various formats (HTML, PDF, Word)

Here are some examples of documents that have been created using Quarto:

- CVs (curriculum vitae) and resumes in PDF
- Research papers for major academic journals
- And more – check the Quarto gallery

Creating your first Quarto document

1. In RStudio, click File → New File → Quarto Document
2. Fill in the title and author
3. Click "Create"
4. Save the document (File → Save As...) with a .qmd extension

To render your document:

1. Click the "Render" button (blue arrow) in the editor toolbar
2. The HTML output will automatically open in your default browser

If you are rendering your document for the first time, it will fail. Look out for a yellow box at the top of your source text that prompts you to install the `rmarkdown` package. Install it (by clicking on the install link), wait for the installation to complete, and then render your document again.

Basic markdown formatting

Quarto uses markdown for text formatting:

Markdown is a simple way to format text that's easy to read and write. The syntax is designed to be intuitive. Quarto's documentation on markdown can be found here.

- **Bold text**: `**bold**`
- *Italic text*: `*italic*`
- Headers: `# Level 1`, `## Level 2`, `### Level 3`
- Lists: Use `-` or `1.` for bullet or numbered lists
- Links: `[text](URL)`
- Images: `![alt text](image.png)`

Code chunks

Code chunks are where you write and execute R code. They keep your code separate from your text while showing both the code and its output.

Code chunks in Quarto start with ```` ```{r} ```` and end with ```` ``` ````:

```{r}
2 + 2
mean(c(1, 2, 3, 4, 5))
```

You can control chunk behavior with options:

- `echo: false` - Hide the code but show results
- `eval: false` - Show code but don't run it
- `warning: false` - Hide warning messages
- `message: false` - Hide messages

For example:

```{r}
#| warning: false
#| message: false

10 / 5
```

Quarto does not run code by default. You need to put R code in code chunks to run them. This is a safety feature to prevent accidental code execution, as well as a means to control the output.

Basic R operations

Now that we have our environment set up, let's try some basic R operations. **Don't forget to use code chunks to evaluate the code below**.

R uses standard mathematical operators. Remember that ^ means "to the power of" and * means multiplication.

```
# Basic arithmetic
5 + 5
10 - 3
4 * 2
8 / 2
2^3 # Exponentiation
```

That's it for now. We will look more into code chunks next week when we focus on statistical operations.

Exercise 3: R packages

What are R packages?

R packages are collections of functions, data, and documentation that extend R's capabilities. Think of them as add-ons or extensions that provide additional functionality beyond what comes with base R. They are essential tools that make R incredibly versatile for different types of analysis.

Think of R packages like apps on your phone - they add new features and capabilities to the base system.

> **Tip**
>
> R comes with several built-in packages (called "base R"), but thousands more are available for specific tasks, from data manipulation to complex statistical analyses.

Installing packages

There are two main ways to install R packages:

You only need to install a package once, but you need to load it every time you start a new R session.

1. Using the `install.packages()` function:

```
# Install a single package
install.packages("readr") # DO NOT PUT THIS IN YOUR QUARTO DOCUMENT

# Install multiple packages at once
install.packages(c("readr", "readxl")) # DO NOT PUT THIS IN YOUR QUARTO DOCUMENT
```

2. Using the RStudio interface:
   - Tools → Install Packages...
   - Type the package name
   - Click "Install"

Do not include `install.packages()` in your Quarto document. This particular function is meant to be run in the R console. Including it in your document causes numerous issues, plus why would you want to install a package every time you render your document?

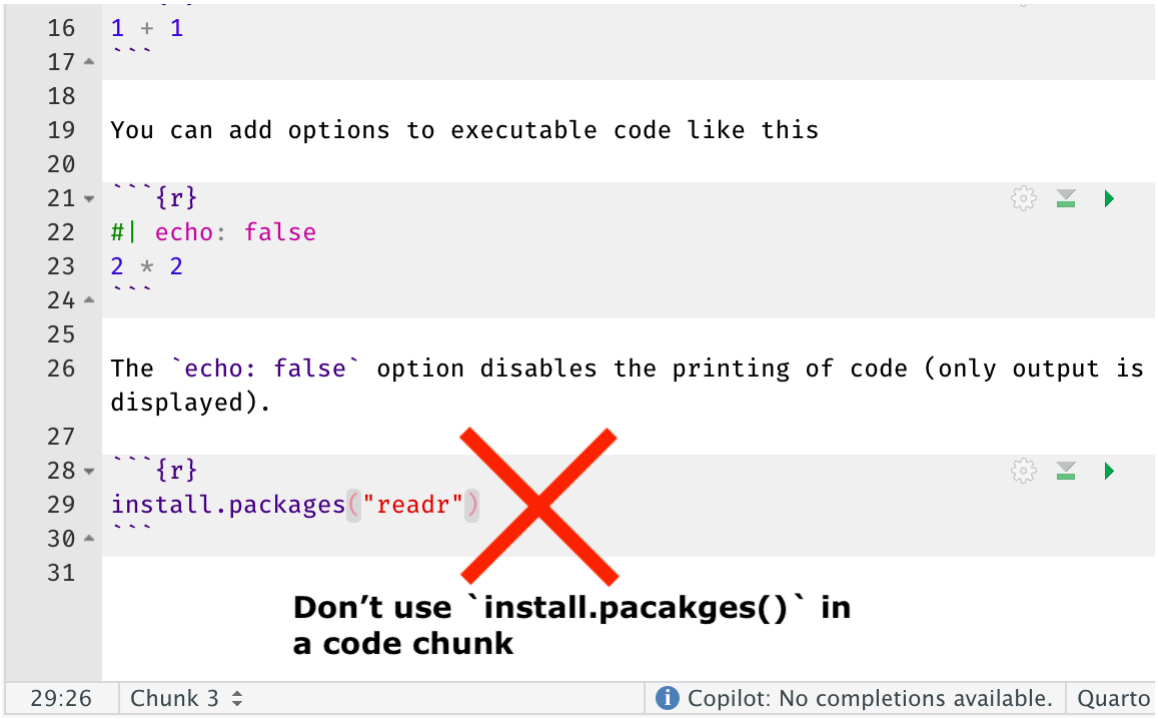

Figure 1: Don't install packages in your Quarto document!

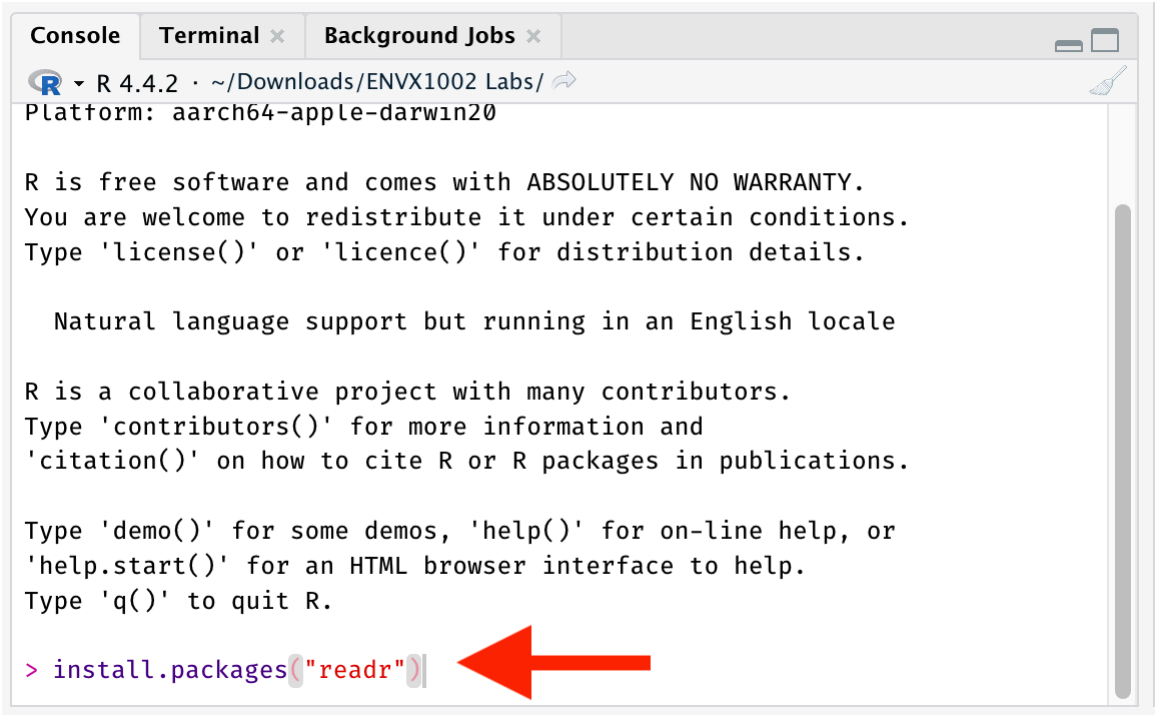

Figure 2: Install packages in the console only. Do it once, and you don't need to do it again.

> **!** Important
>
> You only need to install a package once on your computer. However, you'll need to load it each time you start a new R session.

Loading packages

To use a package in your R session, you need to load it using the `library()` function:

Below, we load the `readr` and `readxl` packages because we need to use the `read_csv()` and `read_excel()` functions to import data and these functions are part of these packages.

```
# Load individual packages
library(readr)
library(readxl)
```

Optional: `pacman`

A more efficient way to handle package management is using the `pacman` package. You can read more about it here. **Please do not use `pacman` unless you have read the documentation and are comfortable with it!**

```
# Load multiple packages at once
pacman::p_load(readr, readxl)
```

> **i** Note
>
> The `::` operator in R allows you to use a function from a package without loading the entire package. For example, `readr::read_csv()` uses the `read_csv` function from the `readr` package.

Exercise 4: importing data

Now that we understand R packages, let's use some specific ones for importing data. We'll use:

- `readr` for importing CSV files
- `readxl` for importing Excel files

Note: CSV files are simple text files that can be opened by many programs. Excel files are more complex but can store multiple sheets and formatting. When Excel opens a CSV file, it treats it like a spreadsheet.

Example data files

Before we begin, download these example files that we'll use throughout this exercise:

- Student Scores (CSV)
- Weather Data (Excel)

Save these files in your project's `data` folder before proceeding.

Understanding file paths

Finding and using files on your computer is a bit like giving directions to your house. Let's break down how R finds your files in a way that's easy to understand:

Think of your project folder as your home base. Relative paths are like giving directions from there: "Go to the kitchen, then the fridge."

What are file paths?

A file path is simply the address of a file on your computer, telling R exactly where to find it. There are two types of file paths:

1. **Absolute paths** are like complete postal addresses:
   - They start from the very root of your computer
   - They work anywhere but are specific to YOUR computer
   - Examples:
     - Windows: `C:/Users/YourName/Documents/ENVX1002/data/student_scores.csv`
     - Mac/Linux: `/Users/YourName/Documents/ENVX1002/data/student_scores.csv`
2. **Relative paths** are like giving directions from where you are now:
   - They start from your project folder (where your .Rproj file is)
   - Much shorter and more convenient
   - Example: `data/student_scores.csv` (meaning "look in the data folder, then find student_scores.csv")

The working directory concept

Your **working directory** is simply the folder R considers as "here" right now:

- When you open an R project, R automatically sets the working directory to that project's folder
- All relative paths are based on this location
- You can check your current location with `getwd()` ("get working directory")

```
# This shows your current "location" in the computer
getwd()
```

How to use paths in practice

If your project structure looks like this:

```
ENVX1002/
├── ENVX1002.Rproj
├── data/
│   ├── student_scores.csv
│   └── weather_data.xlsx
├── images/
│   └── quokka.png
```

```
└── reports/
    └── lab_report.qmd
```

Then from any R code in your project:

- To access student_scores.csv: use `"data/student_scores.csv"`
- To access weather_data.xlsx: use `"data/weather_data.xlsx"`
- To access quokka.png: use `"images/quokka.png"`

It is important to repeat that this works because you created a project in RStudio. If you were to run the same code outside of RStudio it would not work.

Troubleshooting file paths

If R can't find your file, try these steps:

1. Check the spelling and capitalisation (R is case-sensitive!)
2. Make sure the file is actually in the location you think it is
3. Use `list.files("data")` to see all files in your data folder
4. If using an absolute path, double-check it's correct for your computer

> ! Important
>
> When sharing your code with others or moving your project to a different computer, relative paths (like `"data/file.csv"`) will still work, but absolute paths (that include your username and specific computer folders) will break. This is why we recommend always using relative paths in your R projects!

Importing CSV files

CSV files are simple text files where data is separated by commas. They're widely used because they're simple and can be read by most software.

Always check your data after importing it. A quick look at the structure and first few rows can catch common issues early.

To import a CSV file, we use the `read_csv()` function from the `readr` package:

```r
# Using a relative path from the project root
student_data <- read_csv("data/student_scores.csv")

# View the first few rows
head(student_data)

# Get a summary of the data
summary(student_data)
```

> **Tip**
>
> Always try to use relative paths within your R project. This makes your code:
>
> - More portable (works on different computers)
> - Easier to share with others
> - Less likely to break when files move

Importing Excel files

For Excel files, we use the `read_excel()` function from the `readxl` package:

```
# Import an Excel file
weather_data <- read_excel("data/weather_data.xlsx")

# View the first few rows
head(weather_data)

# Look at the structure of the data
str(weather_data)
```

> **Note**
>
> When importing data:
>
> - Always check the first few rows using `head()`
> - Look at the data structure using `str()`
> - Check for any missing values using `summary()`

Common importing issues

Here are some common issues you might encounter when importing data:

Data import problems are common but can usually be fixed by specifying the correct options in your import function.

1. **File path errors**: Ensure you're using the correct path relative to your project directory
2. **Missing values**: R might interpret empty cells differently than expected
3. **Column types**: Sometimes R might guess the wrong data type for columns
4. **Special characters**: Non-ASCII characters might not import correctly

You can handle these issues using additional arguments in the import functions:

```
# Example with more options
student_data <- read_csv("data/student_scores.csv",
  na = c("", "NA", "missing"), # Define missing value codes
  col_types = cols( # Specify column types
```

```
    student_id = col_character(),
    quiz_score = col_double(),
    homework_score = col_double(),
    final_score = col_double()
  )
)
```

Summing up

In this lab, you've learned:

- ☐ Basic setup: Installing R/RStudio and creating projects
- ☐ Creating and formatting Quarto documents
- ☐ Using R packages and basic operations
- ☐ Importing data from CSV and Excel files

Take-home exercises

Exercise 5: Creating a lab report template

Create a Quarto document that will serve as your template for future lab reports. Your template should include:

1. A YAML header with:
   - Your name and student ID
   - The unit of study code (ENVX1002)
   - The current date
   - Output format set to HTML
2. The following sections (using appropriate header levels):
   - Introduction
   - Methods
   - Results
   - Discussion
   - References
3. Include at least one example of each of these formatting elements:
   - Bold text
   - Italic text
   - A bullet point list
   - A numbered list
   - A link to a relevant website
   - An empty R code chunk

Save this template as `lab_report_template.qmd` in your ENVX1002 project folder.

Exercise 6: Data exploration practice

Create a new Quarto document called `data_exploration.qmd` and complete the following tasks:

1. Load the required packages (`readr` and `readxl`)
2. Create a simple data frame with two columns. You may need to use the `data.frame()` function to create this data frame, or you could do this manually in Excel and then import it using one of the `read_*()` functions. The data frame should have the following structure:
   - `site_id`: A to E
   - `temperature`: 15.2, 14.8, 15.6, 14.9, 15.3
3. Use these functions to explore your data frame:
   - `head()` to view the first few rows
   - `str()` to examine the structure and data types
   - `dim()` to check the dimensions (rows and columns)
   - `names()` to see the column names

Remember to add text explanations between your code chunks describing what each function does and what information it provides about your data.

> **Tip**
>
> These exercises will help reinforce the skills you've learned today and create useful resources for future labs. Make sure to render your documents to check that everything works correctly.