1. Installing R

Getting R and an editor onto your own machine

Contents

Install R itself . . . 1
Install an editor . . . 1
A quick tour of the editor . . . 2
Check that it worked . . . 2
Keep each analysis in a Project . . . 2
Packages: the part everyone gets wrong . . . 2
    Try it . . . 3
    Use one function without `library()` . . . 3
Common errors . . . 3
Getting help . . . 4
A working setup, in order . . . 4

This lesson covers a local R setup and the difference between installing and loading a package. You can practise the package example in this browser. Local R is needed to work with your own files and packages that aren't available in the browser.

Install R itself

Download R from CRAN, the Comprehensive R Archive Network:

- Windows: choose **base** and run the `.exe` with the defaults.
- macOS: choose the `.pkg` for Apple silicon (arm64) or Intel.
- Linux: follow CRAN's instructions for your distribution; its default system package may be several versions behind.

> **i** Extra tools for source packages
>
> Windows and macOS usually use ready-built packages. Building a package from source needs Rtools matching your R on Windows, or Xcode Command Line Tools on macOS (`xcode-select --install` in Terminal). Linux often builds from source; Ubuntu's build tools include `r-base-dev`. Return here if an install reports a missing compiler.

Install an editor

R runs in a console. An editor also keeps your script, plots, and help pages in one place. Install R first, then choose a free editor:

- RStudio Desktop is a useful starting choice for this seminar.
- Positron supports both R and Python. It is based on VS Code and left beta in July 2025.

The editor looks for your R installation when it starts.

A quick tour of the editor

In RStudio, the **Source** pane (top left) holds your script; the **Console** (bottom left) runs code. The **Environment** (top right) lists objects, and the bottom-right tabs show files, plots, packages, and help. Positron arranges these differently and calls the Environment pane **Variables**.

Create a script with **File → New File → R Script**. Run a line with **Ctrl + Enter** (Windows/Linux) or **Cmd + Return** (macOS). Save code in the script so you can rerun it. Use `#` comments to explain a choice in the code.

Start with a clean session. Under **Tools → Global Options → General**, untick **Restore .RData into workspace at startup** and set **Save workspace to .RData on exit** to **Never**. Occasionally use **Session → Restart R**, then rerun your script from the top to check that it contains every step.

Check that it worked

Run this in your editor's Console:

```
# R version used by this session
R.version.string
```

Locally, this reports the version you installed, such as `R version 4.6.1 (2026-06-24)` or newer. The cell above runs the browser's R, so its version can differ.

```
# Include this output when reporting a problem
sessionInfo()
```

`sessionInfo()` adds package versions, library paths, and platform details. The browser reports Emscripten/WebAssembly; local R reports your operating system.

Keep each analysis in a Project

A Project keeps data, scripts, and outputs in one folder. Opening it sets R's **working directory**, the folder it reads from and writes to.

In RStudio, choose **File → New Project → New Directory → New Project**. In Positron, create a folder and use **File → Open Folder**.

```
moorfields-seminar/
    moorfields-seminar.Rproj    (created by RStudio)
    data/
        moorfields_baseline.csv
        moorfields_followup.csv
    output/
    analysis.R
```

Keep downloaded data unchanged in `data/` and write results to `output/`. The path `"data/moorfields_baseline.csv"` then works from this Project.

Packages: the part everyone gets wrong

A package adds functions or data to R. Installing and loading are different:

- `install.packages("dplyr")` downloads it to your computer. Run this once in the Console.
- `library(dplyr)` loads it for the current session. Put this in your script.

A missing `library()` call can cause `could not find function "filter"`. Repeated `install.packages()` calls download a package you already have.

```
.libPaths()
```

`.libPaths()` shows where R stores installed packages.

Try it

`palmerpenguins` is installed for this page, but isn't loaded. Run the cell, read the error, then fill the blank to load the package.

```
------

head(penguins)
```

> **Hint**
>
> Use the function that loads an installed package. The package is named `palmerpenguins`.

Solution.

> **Solution**
>
> ```
> library(palmerpenguins)
>
> head(penguins)
> ```
>
> The package is already installed, so it only needs to be loaded.

Use one function without `library()`

`package::function()` calls a function without attaching the package. It still needs to be installed. Part 3 uses this form for `gt::` styling calls.

```
packageVersion("palmerpenguins")

# Use one function without attaching the package
nrow(palmerpenguins::penguins)
```

The prefix also resolves name clashes: `dplyr::filter()` and `stats::filter()` are different functions. If both packages are loaded, the last one loaded supplies the unqualified `filter()`.

Common errors

> **If a package will not install or load**
>
> - `could not find function` or `object not found`: check the spelling and run the code that loads the package or creates the object. R is case-sensitive: `Library(dplyr)` fails.
> - `there is no package called ...`: install it once in the Console.
> - Asked to install from source: choose **No** for the ready-built version unless you have the build tools above.
> - `'lib = "..."' is not writable`: accept R's offer of a personal library.
> - `was built under R version ...`: a warning; the package usually works.
> - `non-zero exit status`: the install failed. Read the first error above it for the missing compiler or system library.

Getting help

A help page lists a function's arguments, defaults, return value, and examples. Start with `?read.csv`; `??"excel"` searches installed help pages.

```
args(read.csv) # Arguments and their defaults
```

`header = TRUE` and `sep = ","` are defaults, so part 2 needn't specify them. Local help opens in the editor's Help pane; `?` may not display a page here.

> **i** More help and package maintenance
>
> When asking for help, include a small reproducible example and `sessionInfo()`; reprex formats code and output for sharing.
> Update packages between projects rather than during a deadline. R releases a new version each April and fixes during the year: moving from 4.5 to 4.6 needs packages reinstalled; 4.6.0 to 4.6.1 keeps them. For longer projects, renv records package versions for later use.

A working setup, in order

Install R and an editor, create the Project folders, and check `R.version.string`. Then install the seminar packages once in your local Console:

```
install.packages(c(
  "gtsummary", "gt", "ggplot2", "plotly", "patchwork",
  "ragg", "svglite", "flextable", "officer", "rvg",
  "readr", "readxl", "writexl", "haven"
))
```

Start your script with the `library()` calls you need. In the browser, packages come from WebAssembly builds rather than CRAN. About half of CRAN isn't available there because of system dependencies. The core examples and optional `flextable`, `officer`, and `rvg` exports use browser-ready packages.

← Lesson overview

2. Downloading and loading data →