

# Workshop on Advanced Sampling Methodologies

Introduction to R Programming

Dr. Gianluca Boo, WorldPop, University of Southampton

2025-09-17

# Agenda

- Introduction to R and RStudio
- Data types and data structures
- Working with scripts and the console
- Importing data
- Saving and closing sessions

# What is R?

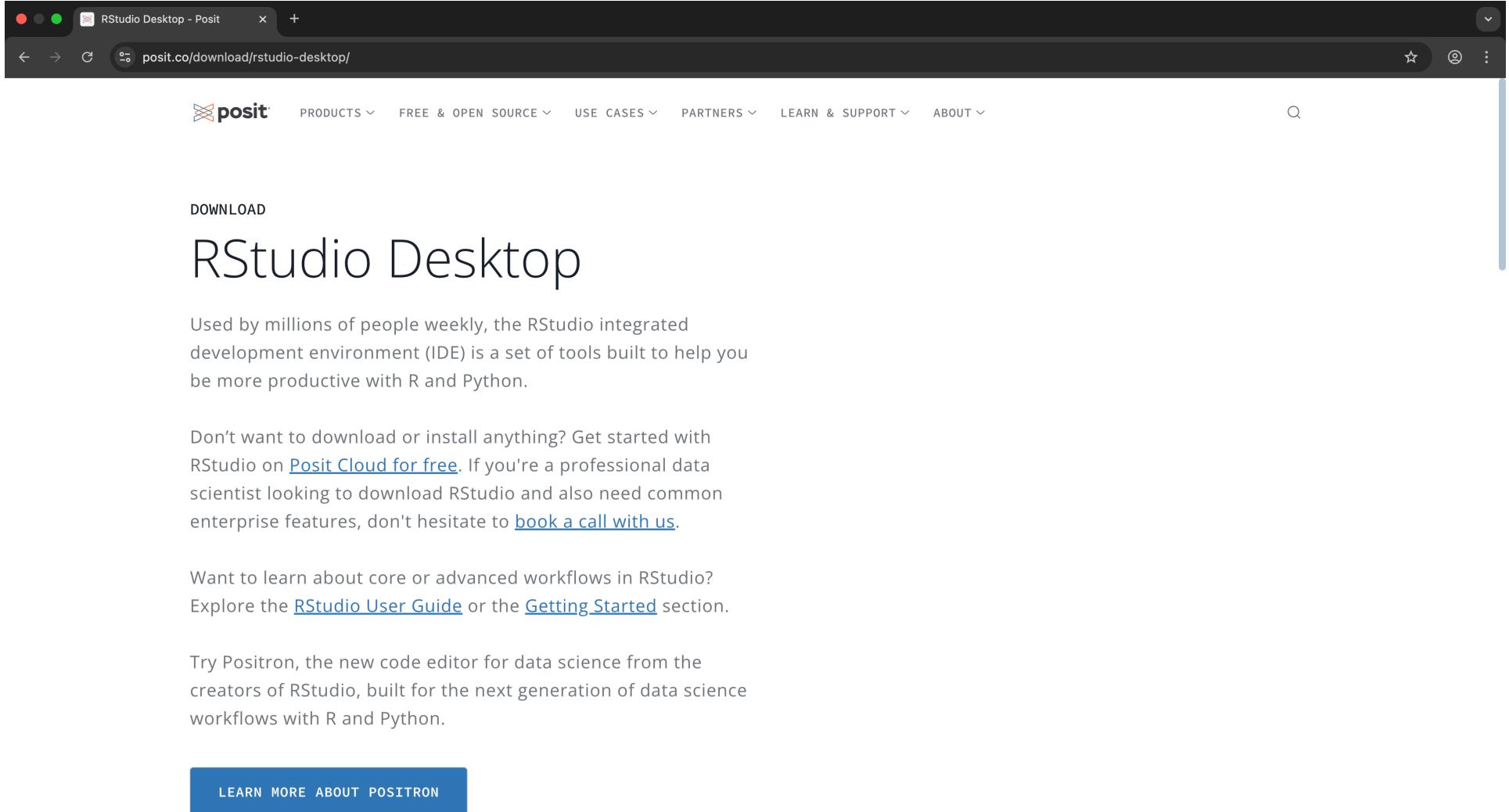
- R is a free, open-source programming language for statistical computing and graphics
- It evolved from the **S language** developed in the 1970s
- Most of R is written in **C** and **Fortran**
- The first stable version was **released in 2000**
- **RStudio** provides a user-friendly interface to R

# Why R?

- Powerful tool for **data analysis and visualization**
- Encourages **reproducibility** and **sharing**
- Thousands of user-created **packages** extend its functionality
- Large and active **community** with extensive support

# Installing R and RStudio

- Download and install **R and RStudio** from posit.co



The screenshot shows a web browser window with the URL [posit.co/download/rstudio-desktop/](https://posit.co/download/rstudio-desktop/). The page has a dark header with the posit logo and navigation links for PRODUCTS, FREE & OPEN SOURCE, USE CASES, PARTNERS, LEARN & SUPPORT, and ABOUT. A search icon is also present. The main content area is titled "DOWNLOAD" and features a large heading "RStudio Desktop". Below the heading, a paragraph explains that RStudio is used by millions weekly and is a development environment for R and Python. It then provides alternatives for those who don't want to download, mentioning Posit Cloud and booking a call. Further down, it encourages learning about workflows through the User Guide and Getting Started sections. Finally, it introduces Positron as a new code editor. At the bottom, there is a blue button labeled "LEARN MORE ABOUT POSITRON".

Used by millions of people weekly, the RStudio integrated development environment (IDE) is a set of tools built to help you be more productive with R and Python.

Don't want to download or install anything? Get started with RStudio on [Posit Cloud for free](#). If you're a professional data scientist looking to download RStudio and also need common enterprise features, don't hesitate to [book a call with us](#).

Want to learn about core or advanced workflows in RStudio? Explore the [RStudio User Guide](#) or the [Getting Started](#) section.

Try Positron, the new code editor for data science from the creators of RStudio, built for the next generation of data science workflows with R and Python.

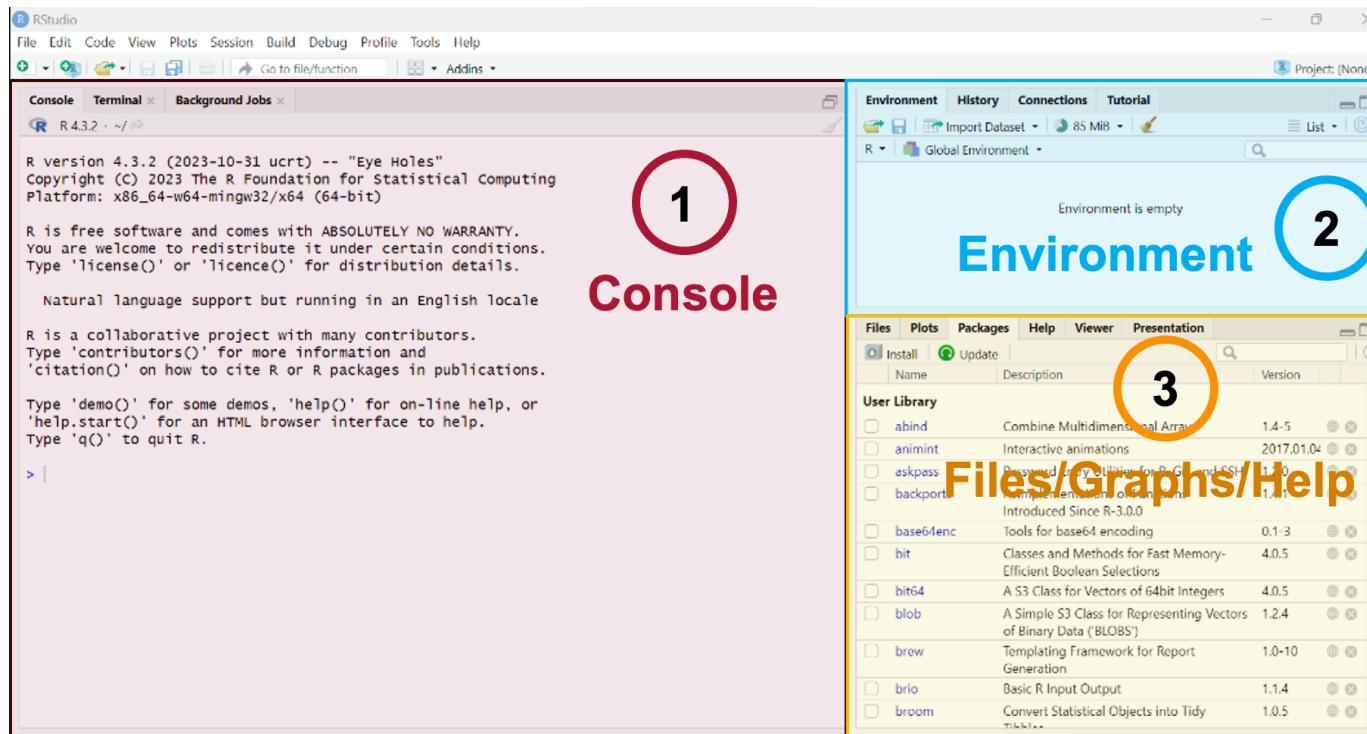
LEARN MORE ABOUT POSITRON



# RStudio Interface

RStudio has four main panes:

- **Source Editor** – write and save code
- **Console** – run commands interactively
- **Environment/History** – see and manage objects
- **Files/Plots/Packages/Help/Viewer** – tools and outputs



# Script Editor vs Console

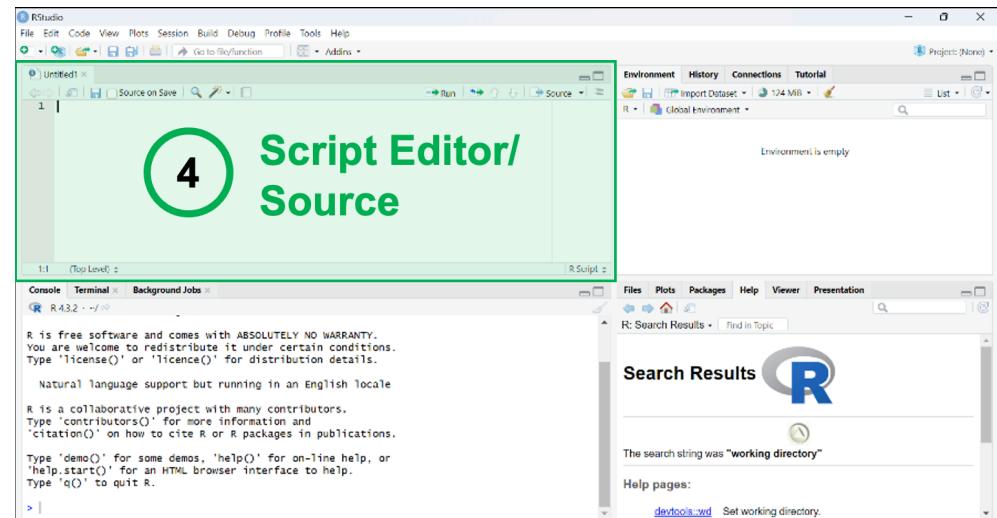
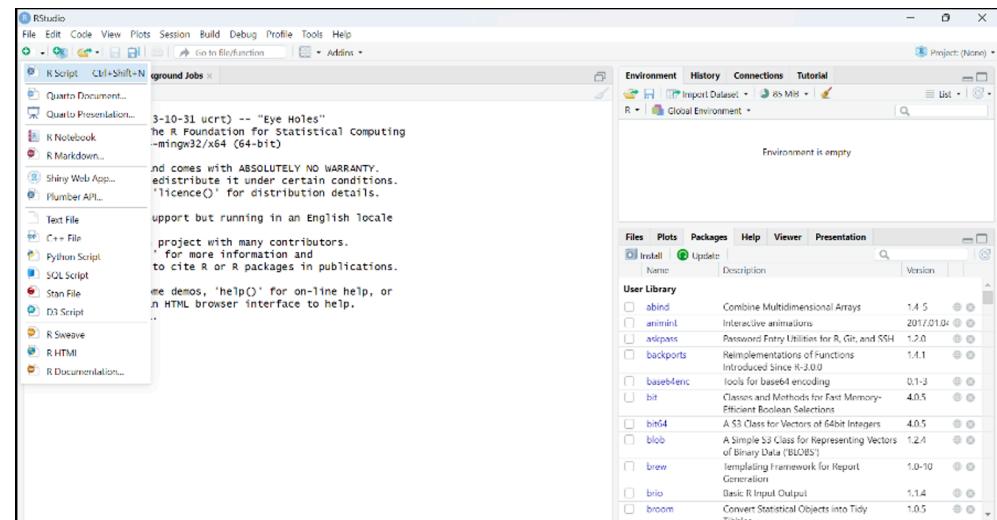
## Script Editor

- Write, edit, and save code
- Organize analysis in reproducible scripts
- Prevent accidental execution before editing

## Console

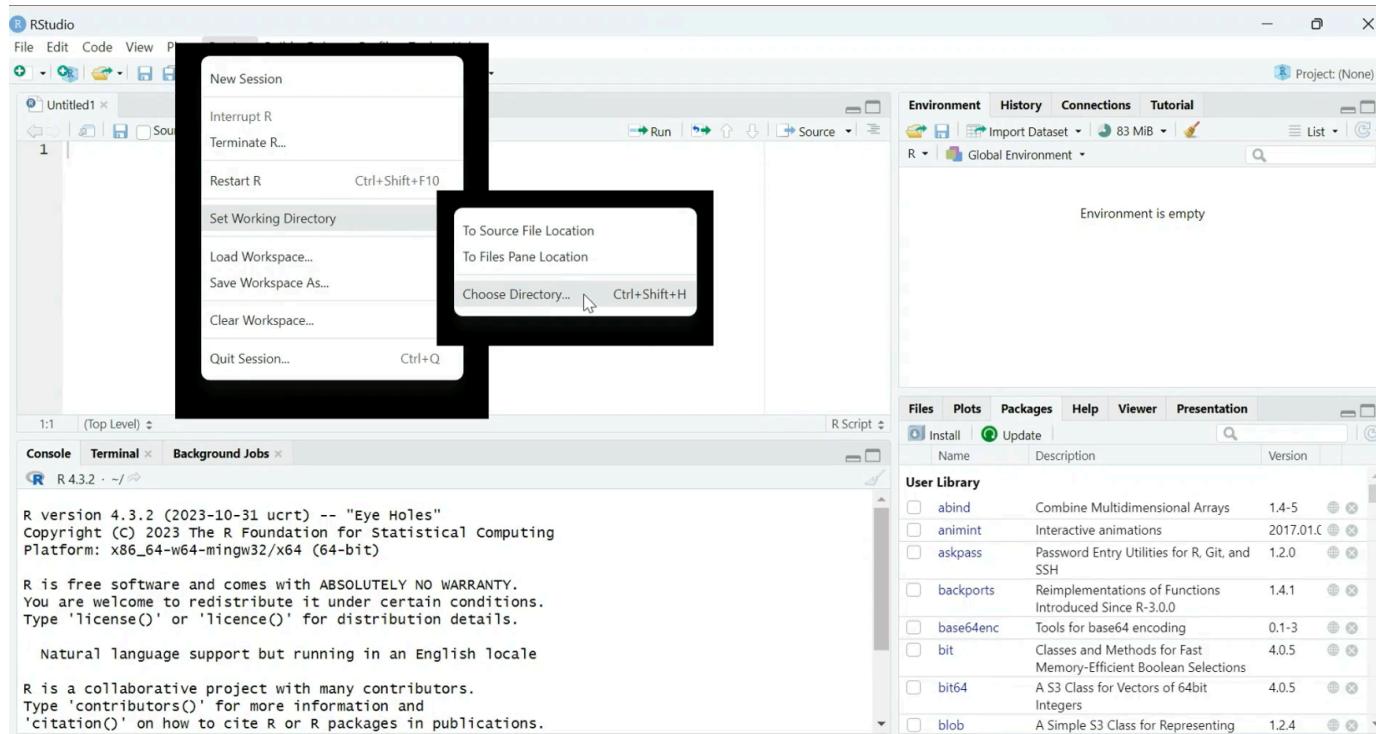
- Executes code immediately
- Use for quick checks and interactive testing
- Output and errors appear here

 Best practice: write in the Script Editor, run via Console.



# Working Directory

- Your working directory is the folder where R looks for and saves files.
- Set with: `setwd("path")`
- Check with: `getwd()`
- Good practice: create a dedicated folder for each project.



Try to run the code in your R Console and in the Script Editor.

# R as a Calculator

- R can perform arithmetic directly

```
1 1 + 2 * 8  
2 a <- log(10)  
3 b <- 5  
4 b + b
```

- Results in Console are prefixed with [1] (index of first element).
- [2], [3], ... appear if output spans multiple lines.

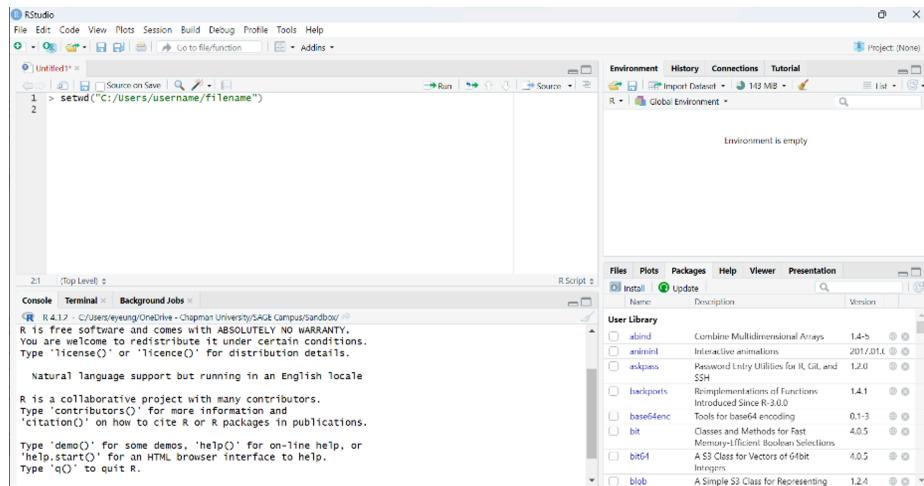
 Try to run the code in your R Console and in the Script Editor.

# R Packages

- Collections of functions, datasets, and documentation.
- Expand R beyond base functions.
- Install from CRAN, GitHub, or other repositories.

## In RStudio

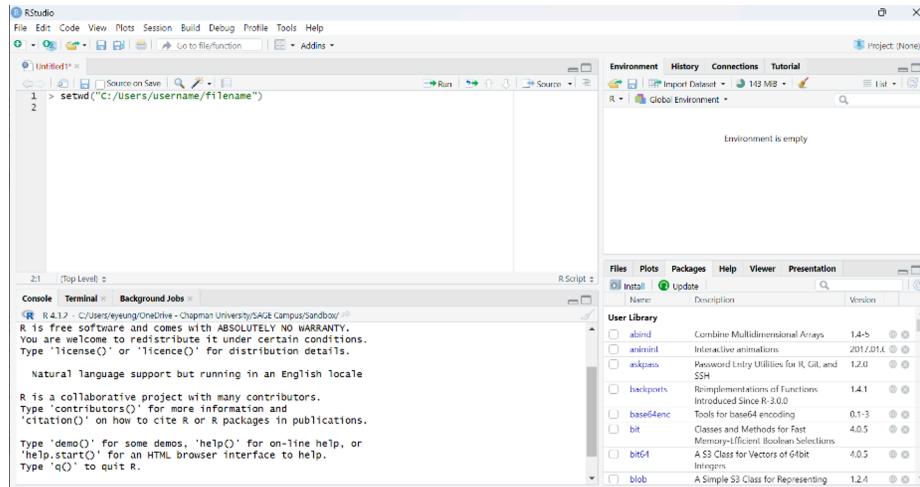
- Use the *Packages* tab to install, update, or remove.
- Or install in the Script Editor



```
1 install.packages("dplyr")
```

# Loading Packages

- Load packages into a session via checkbox in the *Packages* pane.
- Or load packages into a session with:



```
1 library(dplyr)
```

Best practice: only load what you need.

# R Objects

- Everything in R is an **object** (vectors, lists, functions, data frames, etc.).
- Assign values using `<-`.

```
1 x <- 5
2 x + x # returns 10
```

## Naming tips

- Descriptive and concise
- Use underscores `_` or periods `.`
- Avoid very long names
- Avoid existing function names
- No blank spaces

Try to run the code in your Script Editor.

# Data Types

- **Numeric**: numbers, may include decimals.

```
1 w <- c(1.7, 3)
2 x <- c(1, 2, 3) # integers are numeric without decimals
```

- **Character**: text strings.

```
1 y <- c("male", "female")
```

- **Logical**: TRUE/FALSE values.

```
1 y <- c(TRUE, FALSE)
```

- **Factor**: categorical variables.

```
1 gender <- factor(c("boy", "girl", "girl", "boy"))
```

Check type:

```
1 class(object)
2 typeof(object)
```

# Data Structures

- **Vectors**: one-dimensional, homogeneous elements
- **Matrices**: two-dimensional, homogeneous elements
- **Lists**: flexible, heterogeneous elements
- **Arrays**: multi-dimensional, homogeneous elements
- **Data frames**: tabular, columns may differ in type

# Vectors

- The most basic structure in R
- One-dimensional, elements must be the **same type**

```
1 numbers <- c(1, 2, 3, 4, 5)
2 colors <- c("red", "green", "blue")
3 logical_values <- c(TRUE, FALSE, TRUE)
```

Indexing with brackets:

```
1 numbers[1]      # first element
2 colors[2:3]     # second and third elements
```

Vectorized operations apply to all elements:

```
1 numbers * 2      # multiplies each element by 2
```

 Try to run the code in your Script Editor.

# Matrices

- Two-dimensional, **homogeneous elements**
- Create with `matrix()`

```
1 rownames <- c("math", "science", "history", "English", "law")
2 colnames <- c("male", "female")
3 subject_matrix <- matrix(
4   c(10,9,11,13,12,18,17,13,6,5),
5   nrow = 5, ncol = 2, byrow = TRUE,
6   dimnames = list(rownames, colnames)
7 )
```

- Add row: `rbind()`
- Add column: `cbind()`

 Try to add a row to `subject_matrix` for `geography <- c(4,2)` using `rbind` in your Script Editor.

# Matrices to Data Frames

- Convert a matrix into a data frame to **mix numeric and non-numeric types**.

```
1 matrix_stats <- matrix(  
2   c(4015,3980,3756,3333,4000,3000,2000,1000),  
3   nrow = 4, ncol = 2, byrow = TRUE,  
4   dimnames = list(  
5     c("freshmen", "sophomores", "juniors", "seniors"),  
6     c("enrollment", "living on campus"))  
7 )  
8 )  
9  
10 df_stats <- as.data.frame(matrix_stats)  
11 avg.grades <- c("A-","B+","B-","A+")  
12 school_stats <- cbind(df_stats, avg.grades)
```

 Try to run the code in your Script Editor.

# Data Frames

- Tabular structure with columns of different types
- Similar to spreadsheets

```
1 cats <- data.frame(  
2   coat = c("Persian", "black", "tabby"),  
3   weight = c(2.1, 5.0, 3.2),  
4   likes_string = c(TRUE, FALSE, TRUE)  
5 )  
6  
7 animal_sleep <- data.frame(  
8   ranking = c(4, 1, 2, 3),  
9   animal = c("koala", "capybara", "camel", "panda"),  
10  country = c("Australia", "Brazil", "Egypt", "China"),  
11  avg_sleep_hr = c(20, 3, 6, 12)  
12 )
```

 Try to run the code in your Script Editor.

# Lists

- Flexible data structure with elements of **different types**
- Can include vectors, data frames, matrices, or other lists

```
1 my_list <- list(  
2   name = "Alice",  
3   age = 25,  
4   scores = c(90, 85, 92),  
5   passed = TRUE  
6 )  
7  
8 my_list$name  
9 my_list[[2]]
```

 Try to run the code in your Script Editor.

# Arrays

- Extend vectors to two or more dimensions
- All elements must be the same type

```
1 arr <- array(1:12, dim = c(2, 3, 2))
2 arr
3 arr[1, 2, 1] # element at row 1, col 2, slice 1
```

Useful for structured multi-dimensional data. However, this is hardly used in most cases.

 Try to run the code in your Script Editor.

# Importing Data

- Create data manually, but usually import from files

## Common formats

- **Text**: `.txt` (`readLines()` function)
- **Tabular data**: `.csv`, `.tsv` (`read.table()` function or `read_csv()` function from the `readr` package)
- **Excel**: `.xlsx` (`xlsx` package)
- **Google sheets**: (`googlesheets` package)
- **Statistics program**: SPSS, SAS (`haven` package)
- **Databases**: MySQL (`RMySQL` package)

Example:

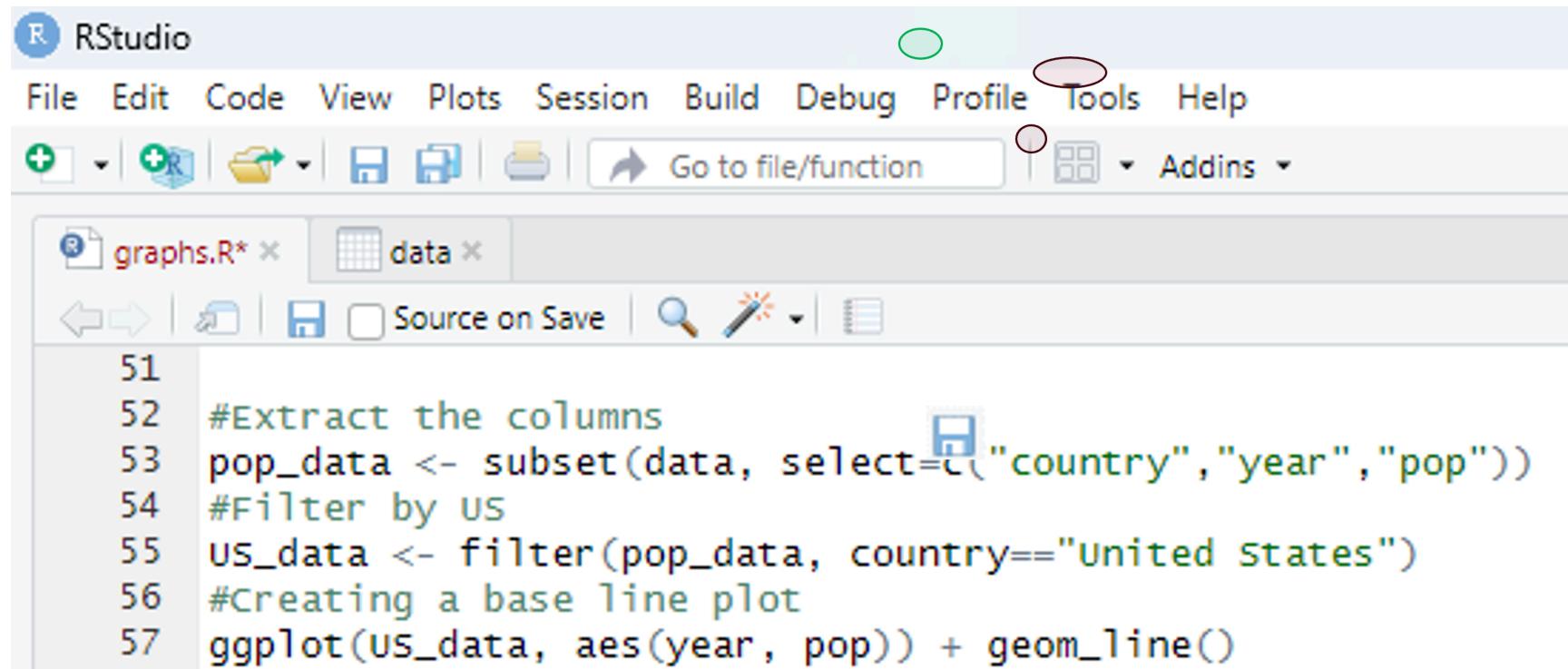
```
1 csv_data <- read.csv("data/world_data.csv")
```

→ Always use forward slashes `/` in file paths.

ℹ Try to run the code in your Script Editor.

# Saving Your Script

- Save all steps of analysis in **.R** scripts
- Add comments with **#**
- Save with **File → Save** or **Ctrl+S / Cmd+S**
- Unsaved scripts show in red with an asterisk \*



The screenshot shows the RStudio IDE. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile (highlighted with a green oval), Tools (highlighted with a pink oval), and Help. Below the menu is a toolbar with various icons. The main workspace shows two files open: "graphs.R\*" and "data". The code editor contains the following R script:

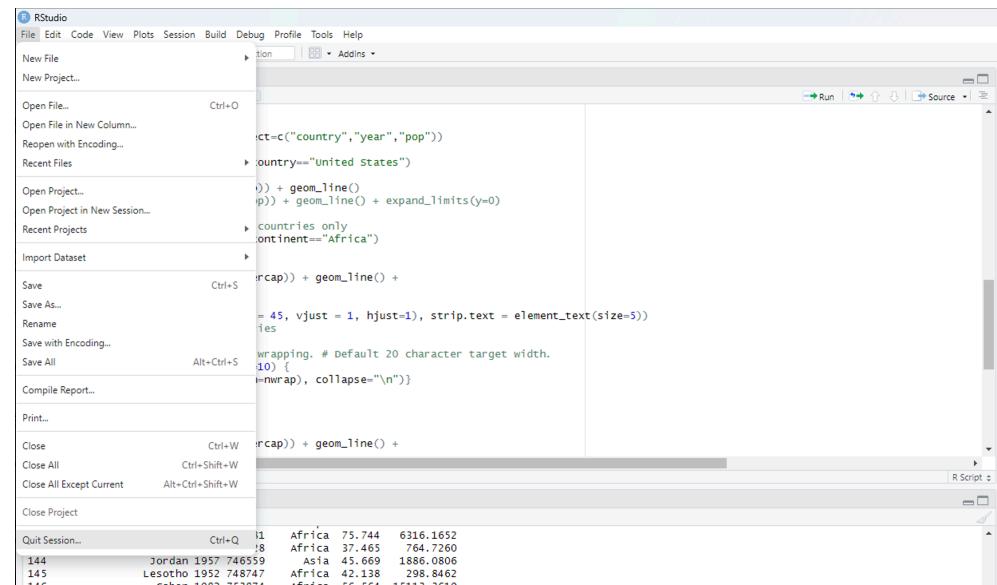
```
51
52 #Extract the columns
53 pop_data <- subset(data, select=c("country","year","pop"))
54 #Filter by US
55 us_data <- filter(pop_data, country=="United States")
56 #Creating a base line plot
57 ggplot(us_data, aes(year, pop)) + geom_line()
```

 Try to save your script.

# Closing an R Session

Options:

- **File → Quit Session**
- `q()` in Console



- ➡ Do **not** save the workspace image. Always start fresh for reproducibility.
- ℹ Try to close and reopen your R Session.

# Take home

- R is a **versatile tool for data analysis and visualization** – free, open-source, and supported by a vast community with thousands of packages.
- RStudio **enhances productivity** – organize, edit, and execute code efficiently using the Script Editor, Console, and built-in panes.
- Understand **R objects, data types, and structures** – Vectors, matrices, lists, data frames, and arrays form the building blocks of R programming.
- **Reproducibility** matters – save scripts, comment your code, and avoid saving workspace images to ensure analyses can be repeated.
- Packages **expand functionality** – install, load, and use packages to extend R's capabilities for importing, manipulating, and visualizing data.

# References

R Core Team. (2025). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. <https://www.R-project.org>

The official reference for R itself, including installation, base functions, and documentation.

RStudio Team. (2025). RStudio: Integrated Development Environment for R. <https://posit.co>

Wickham, H., & Grolemund, G. (2017). R for Data Science: Import, Tidy, Transform, Visualize, and Model Data. O'Reilly Media. <https://r4ds.had.co.nz>

Venables, W. N., Smith, D. M., & the R Core Team. (2023). An Introduction to R. CRAN. <https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf>

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). An Introduction to Statistical Learning with Applications in R (2nd edition). Springer.

# Exercise

Please download the R script with exercises from [GitHub](#) and try to complete it.