

MANCHESTER
1824

The University of Manchester



Introduction to R: Code, Think, Explore

Sol, Amber, Sheezah

Workshop Overview

- 3-hour interactive session
- Friendly for beginners in R
- Mix of live coding, discussion, and setup
- Designed for hands-on learning

Why R?

- Free, open-source, and widely used in research
- Great for data cleaning, statistics, and modelling
- Strong community and support

Session Plan

- **15 min:** Introductions
- **40 min:** R + RStudio setup
- **10 min:** Break
- **10 min:** Quiz
- **40 min:** R basics + Task
- **15 min:** Break
- **20 min:** R basics + Task
- **30 min:** Open discussion + next steps

Introductions

- Who are you, what are your experiences with R or Stata and when/ in what projects have you used it in?

Setting Up R

- Install R: <https://cran.r-project.org>
- R is the language we use to run our analyses (like the math we type on a calculator)
- Install RStudio: <https://posit.co/download/rstudio-desktop>
- Rstudio is what we use to edit R code (like the interface on a calculator)

Installing R



[CRAN](#)
[Mirrors](#)
[What's new?](#)
[Task Views](#)
[Search](#)

[About R](#)
[R Homepage](#)
[The R Journal](#)

[Software](#)
[R Sources](#)
[R Binaries](#)
[Packages](#)
[Other](#)

[Documentation](#)
[Manuals](#)
[FAQs](#)
[Contributed](#)

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows** and **Mac** users most likely want one of these versions of R:

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

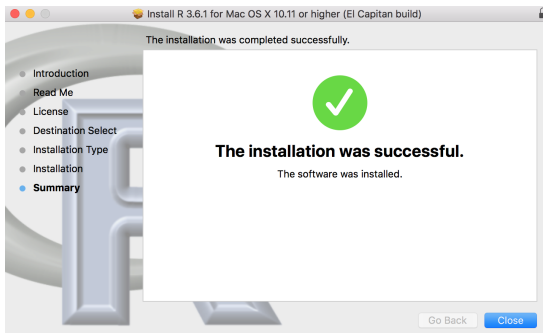
Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2019-07-05, Action of the Toes) [R-3.6.1.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).
- Contributed extension [packages](#)

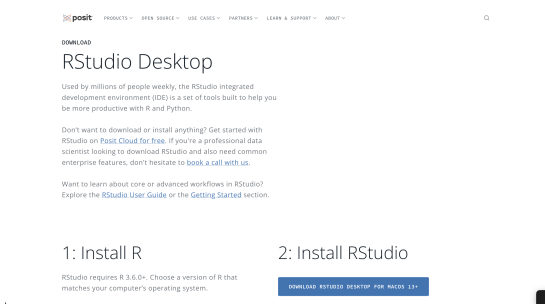
Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

Installing R



Installing Rstudio



The screenshot shows the 'RStudio Desktop' download page on the Posit website. The page has a purple header with the Posit logo and navigation links. The main content area is white and features the title 'RStudio Desktop' in a large, bold font. Below the title, there is a paragraph describing RStudio as an integrated development environment (IDE) used by millions of people weekly. It mentions that RStudio is built to help users be more productive with R and Python. There are two links: 'Posit Cloud for free' and 'book a call with us'. Below this, there is a section titled '1: Install R' which states that RStudio requires R 3.6.0+ and advises choosing a version of R that matches the computer's operating system. To the right of this section is a blue button labeled 'DOWNLOAD RSTUDIO DESKTOP FOR MACOS 11+'. To the right of the button is a small black square. The page also has a search bar in the top right corner and a footer with navigation icons.

posit PRODUCTS ▾ OPEN SOURCE ▾ USE CASES ▾ PARTNERS ▾ LEARN & SUPPORT ▾ ABOUT ▾

Q

DOWNLOAD

RStudio Desktop

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](#).

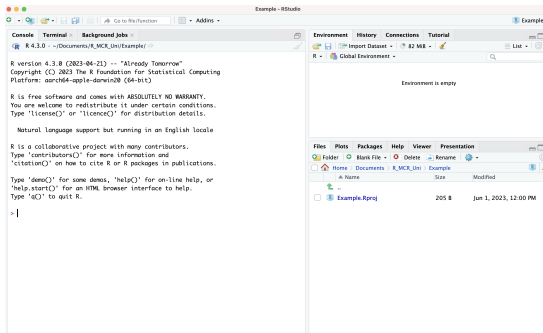
Want to learn about core or advanced workflows in RStudio? Explore the [RStudio User Guide](#) or the [Getting Started](#) section.

1: Install R

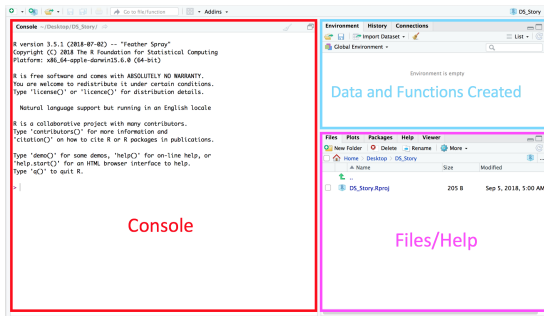
RStudio requires R 3.6.0+. Choose a version of R that matches your computer's operating system.

2: Install RStudio

DOWNLOAD RSTUDIO DESKTOP FOR MACOS 11+



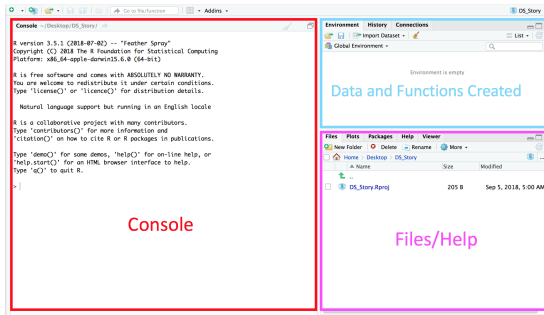
RStudio Interface Overview



The tall red section on the left is the Console and that's where you can type in R code to execute. This code is also called commands or functions.

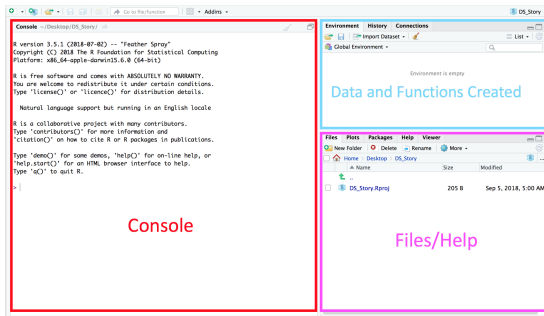
RStudio Interface Overview

In the top right section, there's the Environment tab where you can see the data you are currently working on. At first this section is empty because you have not loaded any data yet.

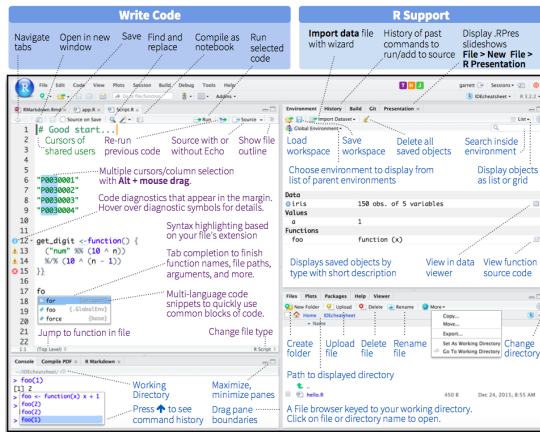


RStudio Interface Overview

In the bottom right section there are tabs to flip through the Files and folder structure of your computer (like in Finder or Explorer), Help information etc.

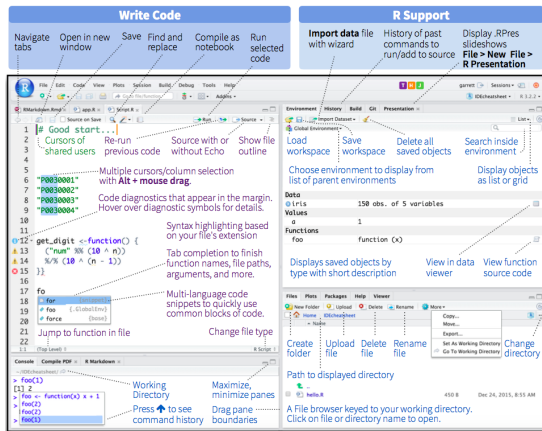


RStudio Window Layout



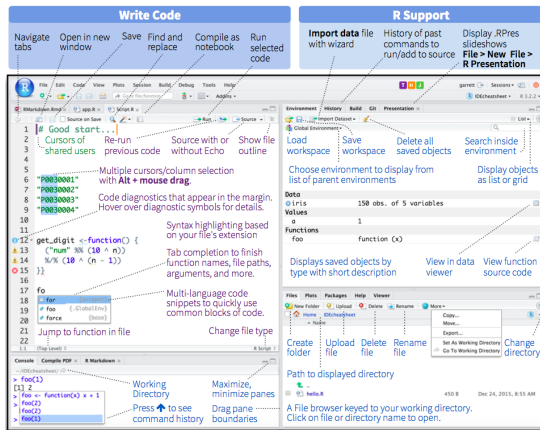
Top Left - Code Editor: for creating scripts to run in pieces or as a whole (like this document!);

RStudio Window Layout



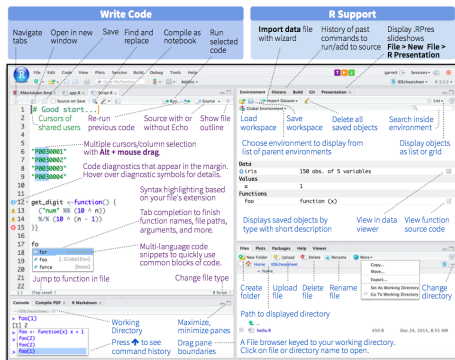
Bottom Left - R Console: you can type R commands and see output;

RStudio Window Layout



Top Right - Environment: lists the objects that you create, such as data sets;

RStudio Window Layout



Bottom Right - Help: find out information about functions and packages. This same pane will have tabs for showing plots that you make, view apps and documents, show files in the folder, and packages used.

Getting Started with R

Let's start using R! We are now going to

- 1 Create an R script to run some R code
- 2 Create a project for your work.

Setting up your working directory

If you want to read or write files on your computer from and to a specific location you will need to set a working directory in R. To set the working directory in R to a specific folder on your computer you will use the following:

```
# On a windows pc, you would set the working directory  
like this
```

```
setwd("C:/Documents/MyR_Project")
```

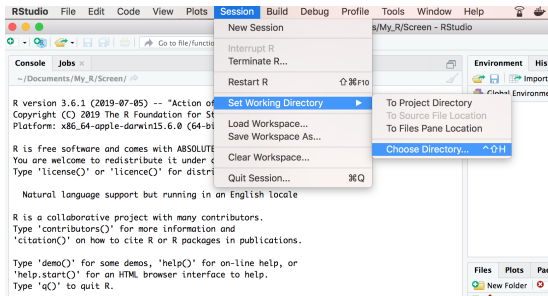
```
# On a mac, you would set the working directory like this
```

```
setwd("~/documents/MyR_Project")
```

Alternative Ways to Set Working Directory

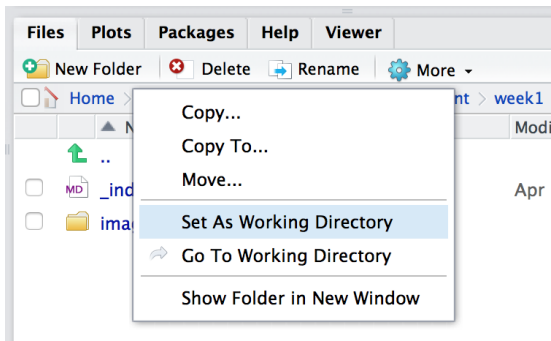
Or, you can also set your working directory by:

- Use the Tools | Change Working Dir... menu (Session | Set Working Directory on a mac).



Alternative Ways to Set Working Directory

- Or, selecting the option from within the Files pane Use the More | Set As Working Directory menu

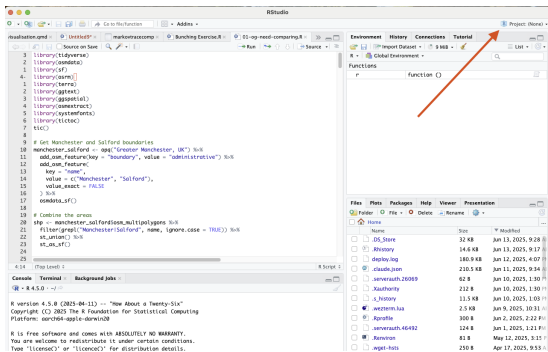


Project-Oriented Workflow

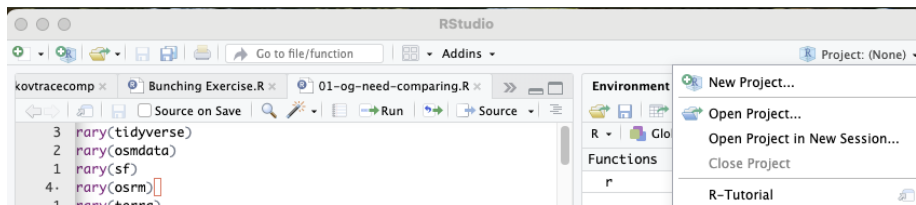
However, you should always start a fresh project (File | New Project...) that will automatically set up your working directory without having to point to it in your script file.

Project-Oriented Workflow

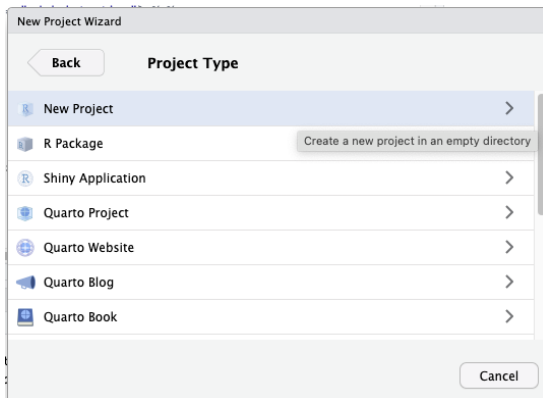
On the very right side of the window is a small blue R, and a drop-down menu. Select New project, then New directory, navigate to the desktop, and name the project My_First_R. This will create a folder with this name on the desktop. This will be your workspace for this project.



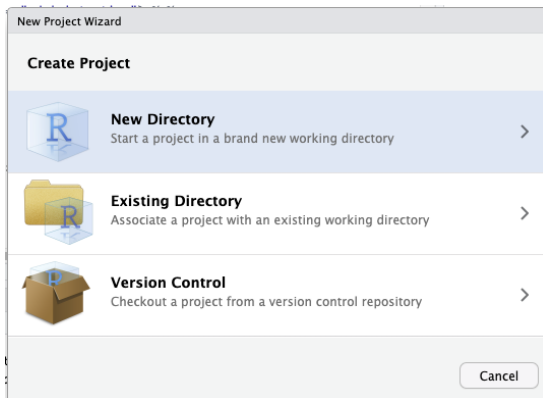
Project-Oriented Workflow



Project-Oriented Workflow




Project-Oriented Workflow



Project-Oriented Workflow

New Project Wizard

[Back](#) **Create New Project**




Directory name:

Create project as subdirectory of:
 [Browse...](#)

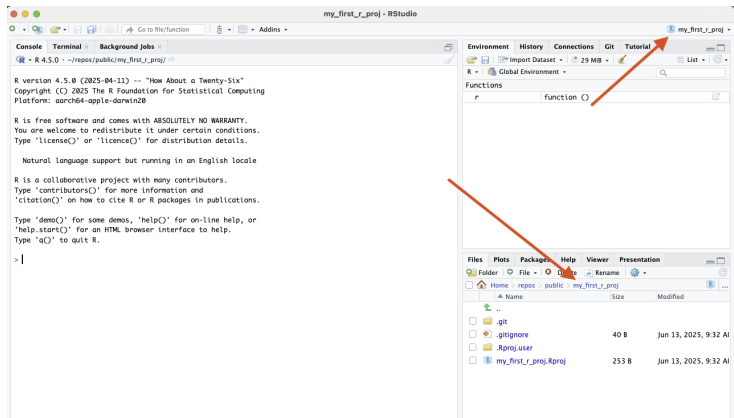
☒ Create a git repository
☐ Use renv with this project

☐ Open in new session

[Create Project](#) [Cancel](#)



Project-Oriented Workflow



Further links: [this blog post](#) to convince yourself that this would be a good habit you should adopt.

R Packages

"In R, the fundamental unit of shareable code is the package."
— Hadley Wickham, *R packages*

R packages are collections of functions code, data sets, documentation and tests developed by the community, that are mostly made available on the Comprehensive R Archive Network, or CRAN, the public clearing house for R packages.

R Packages

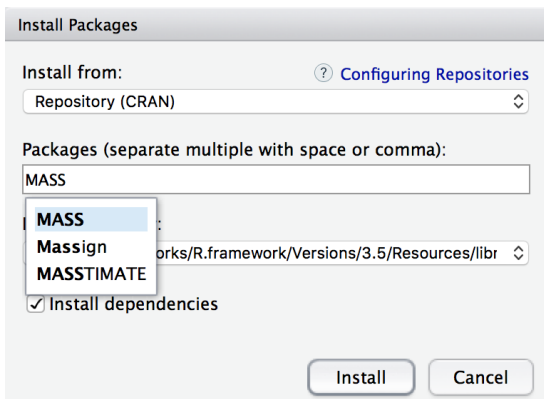
These packages are developed by experts in their fields and currently the CRAN package repository features over 14,000 of them. Many of the analyses that they offer are not even available in any of the standard data analysis software packages, which is one of the reasons that R is so successful.

Installing Packages

When you run R you will automatically upload the `package:base`, which is the system library, i.e. the package where all standard functions are defined. The rest of the so called base packages contain the basic statistical routines.

Installing Packages

Assuming that you are connected to the internet, you can install a package using `install.packages()`. From the RStudio menu, you can do it by selecting **Tools | Install Packages...** and typing the name of the desired package in the dialogue window.



Loading Packages

Once installed, the package will appear in the list of available packages in your Packages pane. To use it you have to load it to the system's search path by simply typing the name of the package as an argument of the `library()` function, or by checking the box next to its name from the Packages pane.

Files

Plots

Packages

Help

Viewer

+

Install

↻

Update

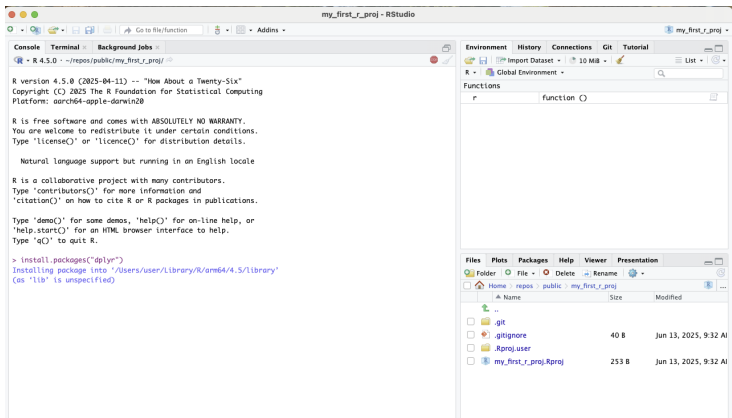
📦

Packrat

	Name	Description	Version	
<input type="checkbox"/>	magrittr	A Forward-Pipe Operator for R	1.5	<input type="button" value="✕"/>
<input type="checkbox"/>	mapproj	Map Projections	1.2.6	<input type="button" value="✕"/>
<input type="checkbox"/>	maps	Draw Geographical Maps	3.3.0	<input type="button" value="✕"/>
<input type="checkbox"/>	maptools	Tools for Handling Spatial Objects	0.9-5	<input type="button" value="✕"/>
<input type="checkbox"/>	markdown	'Markdown' Rendering for R	0.8	<input type="button" value="✕"/>
<input checked="" type="checkbox"/>	MASS	Support Functions and Datasets for Venables and Ripley's MASS	7.3-51.1	<input type="button" value="✕"/>
<input type="checkbox"/>	Matrix	Sparse and Dense Matrix Classes and Methods	1.2-15	<input type="button" value="✕"/>
<input type="checkbox"/>	MatrixModels	Modelling with Sparse And Dense Matrices	0.4-1	<input type="button" value="✕"/>
<input type="checkbox"/>	matrixStats	Functions that Apply to Rows and Columns of Matrices (and to Vectors)	0.54.0	<input type="button" value="✕"/>
<input type="checkbox"/>	mclust	Gaussian Mixture Modelling for Model-Based Clustering	5.4.1	<input type="button" value="✕"/>

Loading Packages

Note: Note that when you can call the dialogue window to install a package from the Packages pane it runs a command in the console for you. Can you figure out the syntax to install a package in R?



The screenshot shows the RStudio interface with the following components:

- Console:** Displays the R version (4.5.0), copyright information, and the output of the `install.packages("dplyr")` command. The command successfully installed the `dplyr` package into the user's library.
- Files Pane:** Shows the project directory structure, including the `.git` folder, `.gitignore` file, `.Rproj.user` folder, and the `my_first_r_proj.Rproj` file.

```
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-darwin28

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.

Natural language support but running in an English locale

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.

> install.packages("dplyr")
Installing package into '/Users/user/Library/R/arm64/4.5/Library'
(as 'lib' is unspecified)
```

Calculate in R

To begin with, we can use R as a calculator.

In your console type in $2 + 2$. Note that you don't have to type the equals sign and that the answer has `[1]` in front. The `[1]` indicates that there is only one number in the answer. If the answer contains more than one number it uses numbering like this to indicate where in the 'group' of numbers each one is.

```
2 + 2
```

You see?! R is like a big calculator!

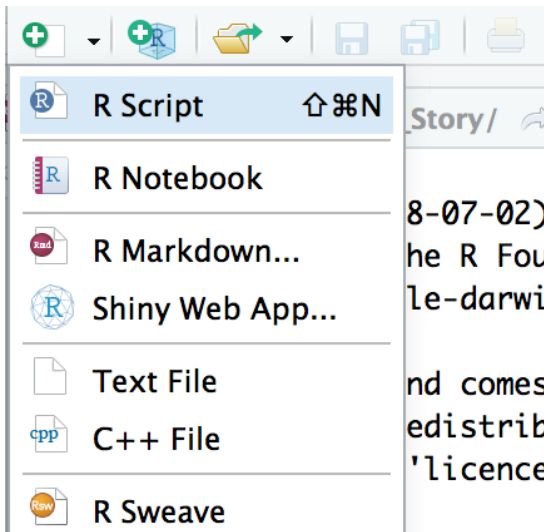
Reproducibility: Save your scripts

Note: The code you type and want to be executed can be saved in scripts and R Markdown files. Scripts ending with `.R` file extension and R Markdown files, which mixes both R code and Markdown code, end with `.Rmd`.

Note: The code that you write just for quick exploration can be written in the console. Code we want to reuse and show off later should be saved as a script.

Creating New Scripts

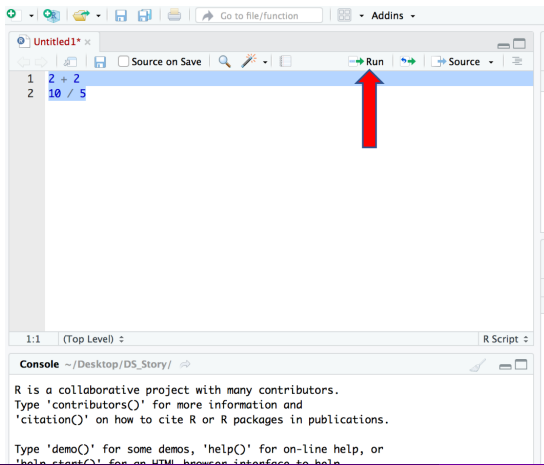
To create a new script go through the menu File | New File | R Script or through the green plus button on the top left.



Running Scripts

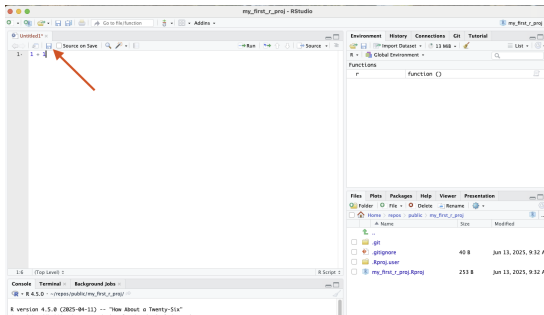
Any code we type in here can be run and executed in the console. Hitting the Run button at the top of the script window will run the line of code on which the cursor is sitting.

To run multiple lines of code, highlight them and click Run.



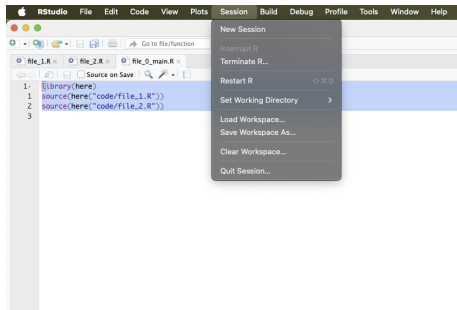
Running Scripts

Tip: Get into the habit of saving your scripts after you create them. Try to save them before running your code in case you write code that makes R crash which sometimes happens



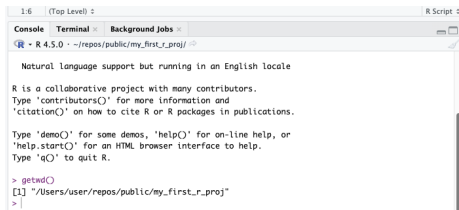
The single most useful R tip

Often, when working with objects in R you may want to start from a new environment, to start from fresh. You may have heard of `'rm(list=ls())'`. In order to ensure your analysis is reproducible, save often and restart R often! This ensures R starts with a blank slate and there are no hidden objects or dependencies clogging up your analysis!



File Paths and Organising Projects in R

The use of R project lies in the ease of organisation



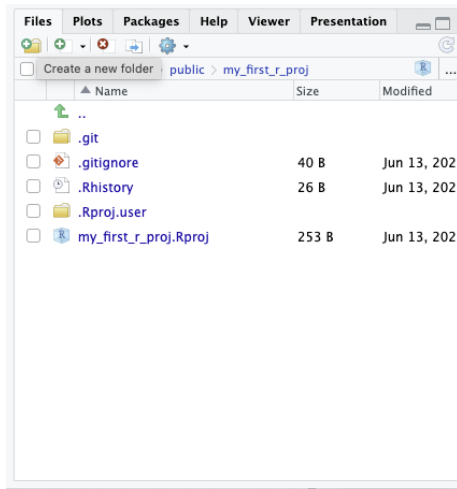
```
1:6 (Top Level) R Script  
Console Terminal Background Jobs  
R • R 4.5.0 • ~/repos/public/my_first_r_proj/  
Natural language support but running in an English locale  
R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to 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.  
> getwd()  
[1] "/Users/user/repos/public/my_first_r_proj"  
> |
```

File Paths and Organising Projects in R

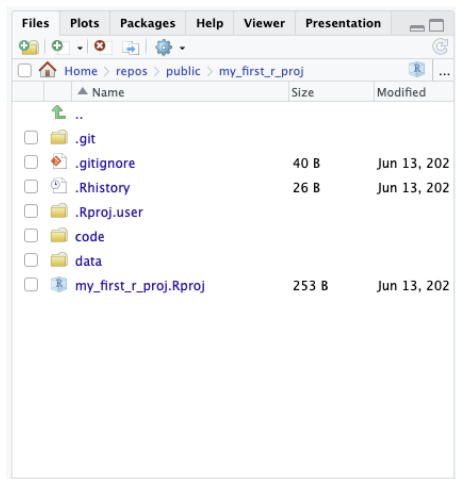
Lets create some folders with specific purposes:

- code - to hold the R files
- data - to hold the data for our analysis
- (optional) plots/output - to hold tables and figures etc

File Paths and Organising Projects in R

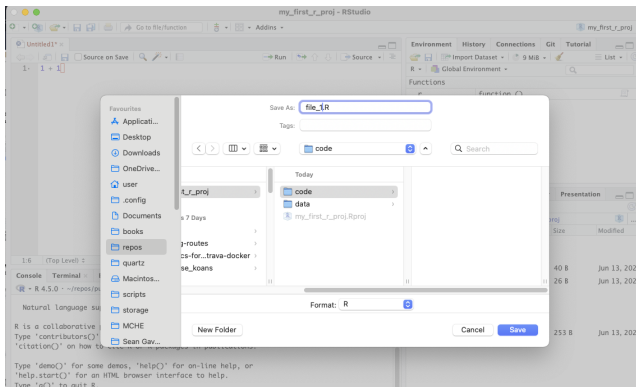


File Paths and Organising Projects in R



File Paths and Organising Projects in R

Now, lets save a file to our code directory



Now, our code is saved in a project folder, and we can source it from outside

File Paths and Organising Projects in R

Let's create a new file that depends on our first file.

file_1.R:

```
answer_1 <- 1 + 1  
answer_2 <- 4*5
```

file_2.R:

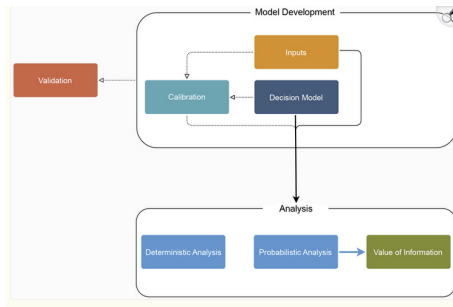
```
print(answer_1 * answer_2)
```

file_0_main.R:

```
library(here)  
source(here("code/file_1.R"))  
source(here("code/file_2.R"))
```

File Paths and Organising Projects in R

You may have seen diagrams like this



Believe it or not, this is essentially what we have just created!

Stretch and refresh!

Quiz Time!

Let's warm up now with something fun to do.

- Go to `mentimeter.com`, or use the app.
- `https://www.menti.com/alk7huvzqpg3`
- Type in the code that's shown upon the screen.
- Test what you know, it's all about using R!

R Basics: Part 1

- Operations
- Objects
- Evaluations
- Exercise

Key Concept

- R is a flexible and powerful tool - far beyond what spreadsheets like Excel can offer.
- As data analysis shifts more towards script-based solutions (like R), it's important to get comfortable with this approach.
- We know it can feel overwhelming at first - the learning curve is real!
- This session is here to make that first step easier and help you begin exploring R at your own pace.

Arithmetic Operators

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/2 is 2

Logical Operators

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x y	x OR y
x & y	x AND y
isTRUE(x)	test if X is TRUE

Basic operations

Let's start with a few simple calculations. These help you get used to the way R responds.

```
1 + 1      # Adds 1 and 1
12 / 4     # Divides 12 by 4
3 * 7      # Multiplies 3 by 7
10 ^ 3     # 10 to the power of 3
```

You'll see output like:

```
[1] 2
```

That `[1]` just means this is the first item in the output. If R prints out lots of numbers, it uses `[x]` to show you where each row starts.

Basic operations

Operator Precedence (Order of Operations): Just like in maths (BODMAS), R follows a specific order when solving expressions:

```
5 * 6 + 2      # Multiplies first, then adds
5 * (6 + 2)    # Adds first, then multiplies
```

Assignment Operator in R

In R, to save values into objects (variables), we use the **assignment operator** `<-`.

This means: "store the value on the right into the object named on the left."

```
x <- 10      # Assign 10 to object x
y <- x + 5    # Assign to y the result of x plus 5
```

Note: In many other programming languages, `=` is used for assignment, but in R, the preferred operator is `<-`.

Remember the `+` sign, by contrast, is an **arithmetic operator** used for addition, not assignment.

Objects in R

R is an **object-based** language. You create and work with *objects*, which are named containers for data.

Think of an object as a labelled box that holds something useful. In R, objects can contain:

- A single number (e.g., `my_num <- 5`)
- A list of numbers (*vector*)
- A table of data (*data frame*)
- A grid of values (*matrix*)
- A custom function

Objects in R

To create an object, give it a name and use the assignment operator `<-` to assign it a value.

Remember, R is case-sensitive - variable names like `Data` and `data` refer to different objects.

We can then use these **objects** going forward rather than the values directly. Operations can be applied to these objects, and objects can be over-written.

Creating and Managing Objects in R

For Example:

```
x <- 48
```

This means “x gets 48”. It stores the number 48 in an object called x. To see what’s inside an object, simply type its name:

```
x  
## [1] 48
```

R remembers this object during your current session and lists it in the Environment tab in RStudio.

Tip: Click on the “Grid” view in the Environment tab to see more details like the object’s type and size.

Objects Can Hold Different Things/ Updating an Object

You can **store** other types of data too. For example, text:

```
y <- "R is cool"
```

This stores a character string. Don't forget the quotation marks, or you'll get an error:

```
y <- R is cool  
Error: unexpected symbol in "y <- R is"
```

You can easily **change** the contents of an object by giving it a new value:

```
y <- 1024
```

Now `y` holds a number instead of text, and its type updates in the Environment.

Working With Objects

Once you've created objects, you can use them in calculations:

```
z <- x + y
z
## [1] 1072
```

This works because both objects are numeric. But if you try this with text, R will complain:

```
my_obj <- "hello"
my_obj2 <- "world!"
my_obj3 <- my_obj + my_obj2
Error in my_obj + my_obj2 : non-numeric argument to
  binary operator
```

Why Can't We Add Text with +?

In R, operations like `+` are designed to work with **numeric data**. If you try to use `+` with **character strings** - such as words or sentences, R will not know how to handle the operation and will return an error.

This error means R expected numbers but received text instead. To combine text values (also called *concatenating strings*), use:

- `paste(x, y)` - joins with a space in between: "Hello World"
- `paste0(x, y)` - joins directly without space: "HelloWorld"

Tip: Use `paste()` when you want spacing, and `paste0()` when you don't.

Exercise

What R code would you write to work these out?

- 1 What is 3×17 ?
- 2 Create an object `d` equal to 10.
- 3 Divide `d` by 5.
- 4 Overwrite `d`, let it now be equal to 20, and print `d` to see the result.
- 5 Overwrite `d` (currently 20) with $4 \times d$, and print `d`.
- 6 Overwrite `d` (currently 80) with $d \times d$, and print the result.

Solutions

- ❶ What is 3×17 ?

```
3 * 17
```

- ❷ Create an object `d` equal to 10.

```
d <- 10
```

- ❸ Divide `d` by 5.

```
d / 5
```

- ❹ Overwrite `d`, let it now be equal to 20, and print `d`.

```
d <- 20  
d
```

- ❺ Overwrite `d` (now 20) with 4 times `d`, and print `d`.

```
d <- d * 4  
d
```


- 7 Overwrite `d` (now 80) with `d` times `d`, and print the result.

```
d <- d * d  
d
```

Common Errors

- One frequent mistake is trying to use an object that hasn't been created:

```
my_obj <- 48  
my_obj4 <- my_obj + no_obj  
Error: object 'no_obj' not found
```

Check the Environment tab-if the object isn't there, R won't know what to do.

- Naming objects is surprisingly hard! Try to keep names short but meaningful. Some good formats:
 - output_summary (snake case – recommended)
 - output.summary
 - outputSummary

Common Errors

Avoid:

- Names starting with numbers or a dot followed by a number
- Special characters like !, &, *
- Using reserved words like TRUE, NA, or built-in functions like data

Seeing and Removing Objects in R

Viewing Objects in Your Environment

- When you've created lots of objects, it can be hard to keep track of them.
- You can use the `ls()` function to list all objects currently in your R environment (or simply the environment pane)

```
ls()
```

Removing Objects from the Environment

- `rm()` can be used to remove specific objects.

```
rm(a)           # removes object a  
rm(x, y)        # removes multiple objects
```

- But we recommend getting in the habit of starting fresh and restarting R! **In Rstudio, use Ctrl+Shift+F10 (Windows and Linux) or Shift+Command+0 (Mac OS)**

Logical Evaluations

Evaluations return TRUE or FALSE

- In R, we can ask questions like “Is one number greater than another?”
- The result is a logical value: either TRUE or FALSE.

Examples:

```
4 > 2      # TRUE - 4 is greater than 2
4 > 5      # FALSE - 4 is not greater than 5
4 == 3     # FALSE - 4 is not equal to 3
4 != 3     # TRUE - 4 is not equal to 3
"dog" == "dog" # TRUE - both are the same string
"dog" == "cat" # FALSE - the strings are different
```

Logical Evaluations

Tips for Beginners:

- Use `==` when checking for equality (not just `=`).
- Logical expressions are often used in filters, if-statements, and data analysis.
- You can use these checks to control your program's behaviour based on conditions.

Logical Evaluations

Storing Evaluation Results

- You can save the result of an evaluation in an object.
- This is useful for making decisions or filtering data later.

Example:

```
b <- 4 < 2      # Stores the result (FALSE) in object b  
b              # Prints FALSE
```

When is this useful?

- To check conditions (e.g. if a person is over a certain age)
- To convert a continuous variable into a binary one (e.g. alive or dead)
- Checking if patients meet criteria for treatment or intervention eligibility

Exercise

Try these!

- 1 Create a new object `A` and set it equal to 3.
- 2 Create another object `res`, let it be the product of `b` (where `b` is equal to 10) and `A`, and output the result.
- 3 Take the average of `b`, `A`, and `res`.
- 4 Is `6.2` equal to `12.4 / 2`?
- 5 Let `m = 84 / 106` and `q = 156 / 3`. Is `m / q` greater than, less than, or equal to 0.0152?

Solutions

- 1 Create a new object A and set it equal to 3.

```
A <- 3
```

- 2 Create another object res, let it be the product of d and A, and output the result.

```
res <- d * A  
res
```

- 3 Take the average of d, A, and res.

```
(d + A + res) / 3  
# mean(c(d, A, res))
```

- 4 Is 6.2 equal to 12.4 / 2?

```
6.2 == 12.4 / 2
```

- ⑥ Let $m = 84 / 106$ and $q = 156 / 3$. Is m / q greater than, less than, or equal to 0.0152?

```
m <- 84 / 106
q <- 156 / 3
m / q > 0.0152
m / q < 0.0152
m / q == 0.0152
```

Stretch and refresh!

R Basics: Part 2

- Object classes
- Object structures
- Loading a dataset in R

R Basics: Object Classes and Structures

Goal: Understand what type of data (*class*) you're working with, and how data is organised (*structure*).

So far, we've worked with single values (like 1 or 2). But often, objects in R hold multiple values. These could be:

- Heights of several children (**numbers**)
- Names of the children (**text**)
- Whether each child is tall or not (**TRUE/FALSE**)

We will learn:

- What *class* an object is
- How objects behave depending on their class

R Basics: What is an Object Class?

Each object in R has a **class**, which describes the type of data it holds.

- **numeric** – numbers, e.g. 1.45, 3.2
- **integer** – whole numbers, e.g. 4, 22
- **character** – text, e.g. "Alice"
- **factor** – categories, e.g. "Male", "Female"
- **logical** – TRUE or FALSE

R Basics: What is an Object Class?

To check the class of an object, use the `class()` function:

```
# Try these below - what are the classes?
```

```
height <- c(1.38, 1.45, 1.21)
class(height)
```

```
names <- c("Alice", "Bob")
class(names)
```

```
sex <- factor(c("F", "M"))
class(sex)
```

```
tall <- height > 1.4
class(tall)
```

R Basics: What is an Object Class?

To check the class of an object, use the `class()` function:

```
# the answers are....
```

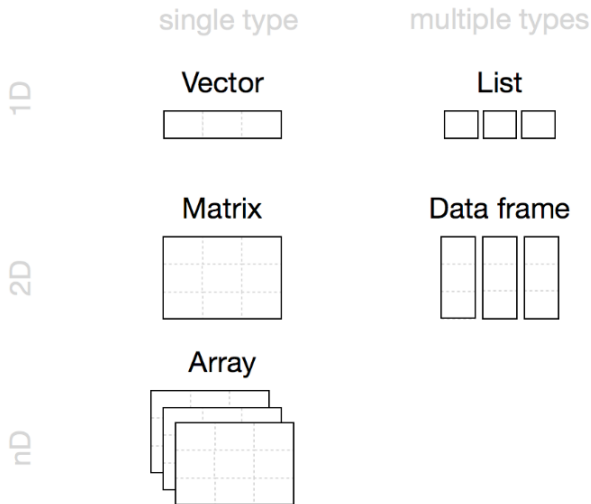
```
height <- c(1.38, 1.45, 1.21)
class(height)           #numeric
```

```
names <- c("Alice", "Bob")
class(names)            #character
```

```
sex <- factor(c("F", "M"))
class(sex)              #factor
```

```
tall <- height > 1.4
class(tall)             #logical
```

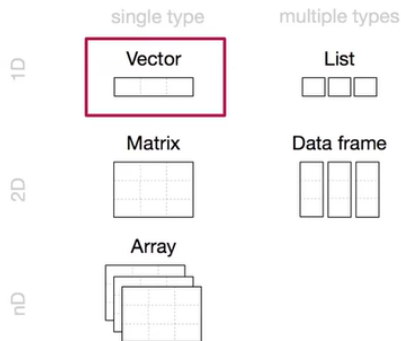

R Basics: What are data structures?



Data Structures

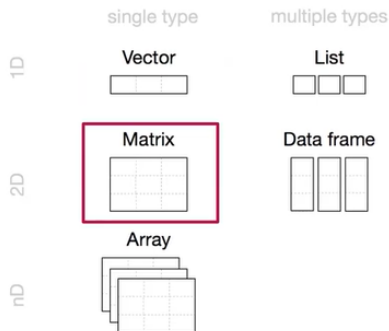
R Basics: Vectors

- One-dimensional arrays
- One type of data (numeric, character, logical...)
- Example: `c(1, 2, 3)`



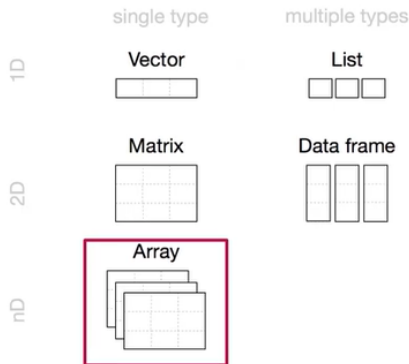
R Basics: Matrices

- Two-dimensional arrays
- One type of data (numeric, character, logical...)
- Example: `matrix(1:9, nrow=3, ncol=3)`



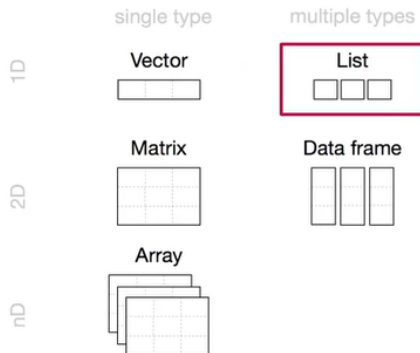
R Basics: Arrays

- Multi-dimensional arrays
- One type of data (numeric, character, logical...)
- Example: `array(1:27, dim= c(3, 3, 3))`



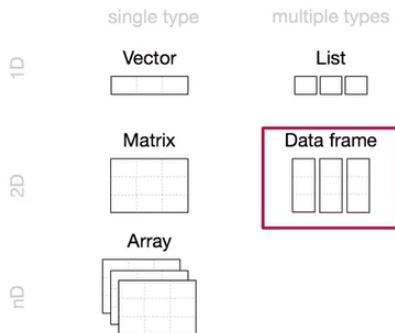
R Basics: Lists

- One-dimensional
- Different data types
- Example: `list(1, "a", TRUE)`



R Basics: Data Frames

- Two-dimensional
- Different data types
- Example: `data.frame (x=1:3, y=c("a", "b", "c"))`



R Basics: How to Create Vectors

We can make vectors (multiple values in one object) using the `c()` function:

```
c(1.2, 2.3, 3.4)      # combines values into one object
```

?c gives help for the combine function. Try it in the console!

R Basics: Operations on Vectors

You can do operations on vectors too:

Add values:

```
c(1, 2, 3) + 1  
c(1, 2, 3) + c(1, 2, 3)  
c(1, 2, 3, 4) + c(1, 0, 1)
```

#Try it in the console!

R Basics: Operations on Vectors

You can do operations on vectors too:

Add values:

```
c(1, 2, 3) + 1
```

```
#2 3 4
```

```
c(1, 2, 3) + c(1, 2, 3)
```

```
#2 4 6
```

```
c(1, 2, 3, 4) + c(1, 0, 1)
```

```
#2 2 4 5
```

```
#what did you notice about the last code?
```

Multiply:

```
heightft <- height * 3.28
```

R Basics: How to Create a Matrix

We can make a matrix using the `matrix()` function:

```
# A 2x3 matrix
```

```
m <- matrix(1:6, nrow = 2, ncol = 3)  
print(m)
```

```
#Try it in the console
```

R Basics: How to Create an Array

We can make an Array using the `array()` function:

```
# Create a 3x3x2 array
```

```
arr <- array(1:18, dim = c(3, 3, 2))  
print(arr)
```

```
#Try it in the console
```

Excerice time!

```
# Create a vector called 'odds' with the numbers  
  1,3,5,7,9.  
  
# Show what class odds is.  
  
# Evaluate which numbers in the odds vector are greater  
  than 4.
```

Solution

```
# Create a vector called 'odds' with the numbers  
odds <- c(1, 3, 5, 7, 9)  
  
# Show what class odds is.  
class(odds) #numeric  
  
# Evaluate which numbers in the odds vector are greater  
than 4.  
odds > 4 #FALSE FALSE TRUE TRUE TRUE
```

R Basics: Loading in a dataset

Base R loads in with some base datasets, packages, functions etc, one of these is the iris dataset

```
data("iris")  
data <- iris
```

Loading this in, the environment tab might show us what it contains, but lets double check this

R Basics: Loading in a dataset

Base R loads in with some base datasets, packages, functions etc, one of these is the iris dataset

```
data("iris")  
data <- iris
```

Loading this in, the environment tab might show us what it contains, but lets double check this

```
colnames(data)  
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.  
      Width"  
[5] "Species"
```

How do we select one of these variables?

R Basics: Loading in a dataset

Base R loads in with some base datasets, packages, functions etc, one of these is the iris dataset

```
data("iris")  
data <- iris
```

Loading this in, the environment tab might show us what it contains, but lets double check this

```
colnames(data)  
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.  
      Width"  
[5] "Species"
```

How do we select one of these variables? With the \$!

```
data$Sepal.Length
```


R Basics: Loading in a dataset

Base R loads in with some base datasets, packages, functions etc, one of these is the iris dataset

```
data("iris")  
data <- iris
```

Loading this in, the environment tab might show us what it contains, but lets double check this

```
colnames(data)  
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.  
      Width"  
[5] "Species"
```

How do we select one of these variables? With the \$!

```
data$Sepal.Length
```

How can we check what class this variable is?

R Basics: Loading in a dataset

Base R loads in with some base datasets, packages, functions etc, one of these is the iris dataset

```
data("iris")  
data <- iris
```

Loading this in, the environment tab might show us what it contains, but lets double check this

```
colnames(data)  
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.  
Width"  
[5] "Species"
```

How do we select one of these variables? With the \$!

```
data$Sepal.Length
```

How can we check what class this variable is?

```
class(data$Sepal.Length)  
[1] "numeric"
```

Final Challenge

Install the cowsay package, load it, and use the 'say' function to write something memorable

Hint 1: you will need to think back to Sol's slides on how to load a package

Hint 2: Ask R to help you understand the function!

Show us what you've done in mentimeter

Solution

Install the cowsay package, load it, and test it:

```
install.packages("cowsay")
library(cowsay)
say("I've installed cowsay!")

cowsay::say("Hello")  # qualified names

#Read the package docs, what else can it do?

vignette("cowsay")
?cowsay::say
help(package="cowsay")

say("why did the chicken cross the road", "chicken")
```

Open Discussion

- Dates for the next session?
- Feedback on this session?
- What would you like to learn next session?

Thank you!

Further Resources:

- *R for Data Science (2e)* – An accessible and online book for the basics of R
- *What They Forgot to Teach You About R* – Another online book about the stuff you need to know about R beyond data analysis, including workflow, environments, and debugging.
- *Tidyverse Koans* – A series of exercises designed to teach and test tidyverse tools (like `dplyr`, `ggplot2`, and `purrr`) libraries