StartR for Psychology at Royal Holloway

Luke Kendrick

2025-04-09

Contents

Chapter 1

Welcome to StartR

Figure 1.1:

StartR is a short course for someone who wants a beginner's guide to using R and RStudio. This resource is suitable for anyone who wants a gentle self-paced introduction.

Figure 1.2: **R** is not just the sound a pirate makes, but it is also a programming language.

1.1 What is R and RStudio?

R is a programming language for statistical computing and data visualisation (see https://www.r-project.org/about.html). As with any language, you could write a set of instructions or operations for someone to follow.

For example, I could ask you to:

```
`stand up` and then,
`walk towards the window` and then,
`return to your seat`
```

I hope you enjoyed looking out of the window for a moment there. We will come back to this later, but the basic logic behind code is writing a series of instructions.

When using R, you'll be giving instructions to our computer to follow and complete.

R is not the same thing as RStudio. RStudio is an integrated development environment (IDE). It provides a user-friendly interface for you to add your instructions and to view output.

Think of the difference between R and RStudio in the same way you would a car dashboard and an engine. The engine of the car works in the background and unless you're a mechanic, it will just work away in the background without much notice. Being seated in the car with the dashboard on the other hand is where you direct or drive your car. RStudio is how you will drive whilst R (the coding language under the hood) is the engine underneath. This book will show you how to drive and to understand how the engine works, but it is really not necessary to become a mechanic, just a good driver!

1.2 Installing R and RStudio

With installation you will need both the car dashboard **and** the engine. You should install two things:

- R (the engine)
- RStudio (the car dashboard)

You can install both of these here.

1.3 Opening RStudio

Once you have installed both of these, you should only open **RStudio**.

Find RStudio on your computer and open it. The RStudio logo is shown in Figure 1.3. A frequent mistake is opening the engine `R`, but you should only use RStudio.

Figure 1.3: Look out for the RStudio logo.

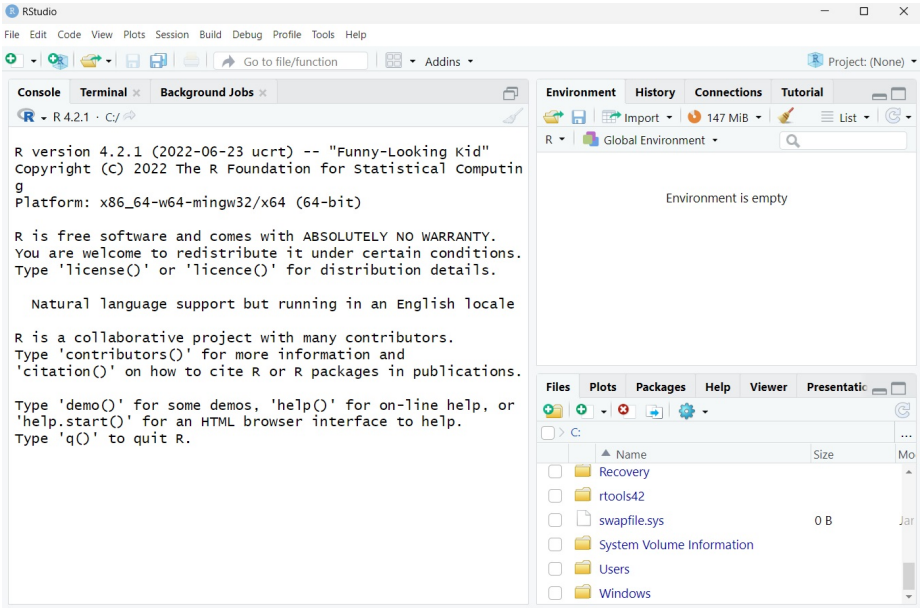

Figure 1.4: RStudio when first opened.

When you open RStudio it should look like the screenshot in Figure 1.4. Notice that there are three main panels.

Before you really dive in to getting RStudio to work, you will need to open a new script. To do this, look for the top left button which looks like a blank white page with a green/white `plus` symbol, just like in Figure 1.5 below.

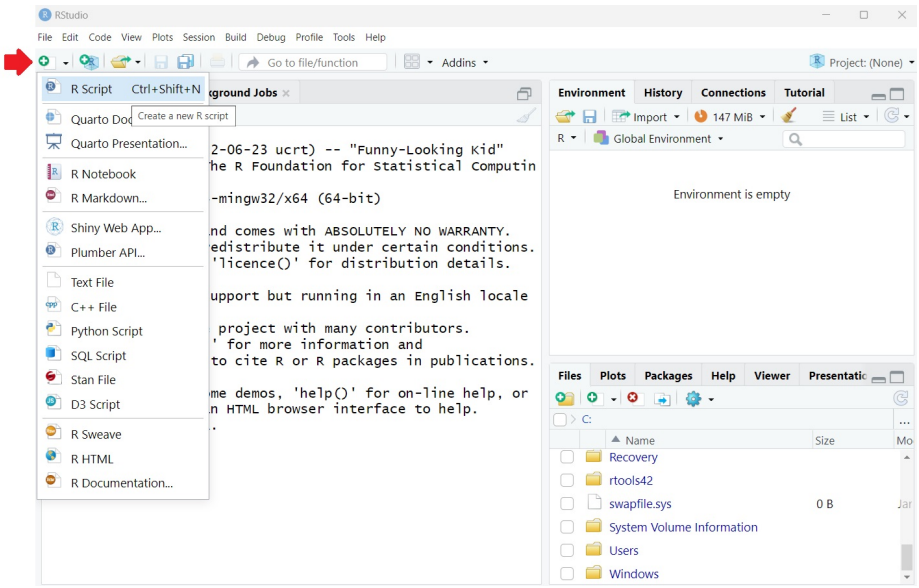

Figure 1.5: Open a new R Script.

Once you open the new script you should now see four panels just like Figure 1.6.

Another way to open a new script is to use the menu at the top of the RStudio window: `File > New File > R Script`.

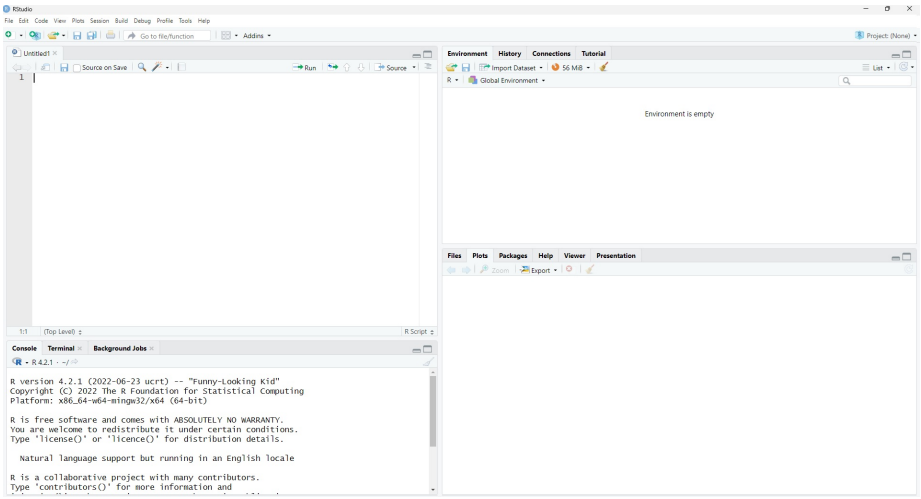

Figure 1.6: You should see four panels now which means you are ready to start working with RStudio.

1.4 A Quick Tour of RStudio

Now it is time to learn about the four panels. Let's go through each one by one. These four panels are shown in Figure 1.7 with a description of each below.

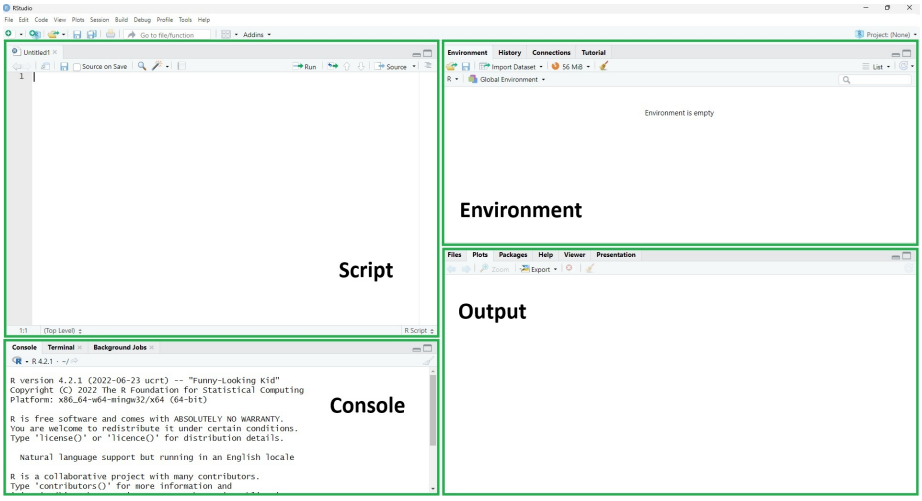

Figure 1.7: The four panels in RStudio.

Script Panel

This is the top left panel. This is an editable space where you will be able to type and edit any code you use. The good news is you can save an R script so you can use it again in the future.

Think of it as an input panel.

Console Panel

The bottom left panel is the console. Whilst you can technically type in any code here, I'd advise you not to as you wont be able to save any code. The console is where you can view any output messages. Essentially, once you type in and run your code in the script, RStudio will respond to you via this console panel.

Think of it as an output panel.

Environment Panel

This is the top right panel. This is a space where RStudio will hold any `objects`. Essentially this is a bit like a memory space where RStudio will keep hold of any important information you ask it to.

Think of it as a memory panel.

Output Panel

The bottom right panel is a bit trickier to name. I will call it the output panel but ultimately it can do a few jobs. You might notice different tabs. `Plots` is where we will see any data visualisation outputs, or plots. `Packages` provides a list of our packages (more on this later). `Help` is also a useful tab as it is a space where you read about R Documentation (instructions how to use bits of code). There is also a `files` tab which can allow you to navigate files on your computer.

1.5 Chapter Summary

At the end of this chapter you should have installed R and RStudio. You should have also opened RStudio and created a new script so that you can see four panels.

Take a few moments to have a little look around these four panels so that you are familiar with what they look like before the next chapter.

Chapter 2

Putting R to Work

In this chapter you will start using code and putting RStudio to work! We will begin with very basic code so that you become familiar with the layout and how entering code works.

2.1 What Day is it Today?

Let's begin with something really simple.

When RStudio is open you want to make sure you have four panels, with the top left panel (script) open and ready. This is where you can type your code.

Type the code below into your script now.

```
date()
```

Once you have typed in your code, you are ready to run it. This means you will ask RStudio to tell you today's date.

To run the code you can leave your cursor at the end of the line of code and use your keyboard to press `Control` + `Enter` or `Command` + `Enter` on a Mac. You might also notice on the top right of your script panel there is a button called `run`. This will also run the line of code but I'd recommend getting used to using the keyboard.

Once you have run the line of code, take a look at the **console** (bottom left panel). If it has worked you should see today's date:

```
date()
#> [1] "Wed Apr  9 14:07:58 2025"
```

Congratulations! You have entered the world of coding.

Well, what else can RStudio do? It can be helpful to also think of the script panel as a very fancy calculator. With this in mind, let's try out some maths sums. Run the following code:

```
20+80
```

If you ran it successfully you should see the answer appear in the **console** (bottom left panel).

Next, try a really complex sum:

```
12345*54321
```

Note that we need to use `*` an asterisk to signify this is a multiplication sum. Run the line of code and check out the answer. In case you needed to know the answer to 12345 x 54321, RStudio is there for you!

Hopefully now you are starting to see the very basics of how you should input code into RStudio and where to check to see the output.

2.2 Functions and Arguments: A Piece of Pi

In the previous section, you ran a bit of code which is a **function** called `date()`. In R, a **function** is a bit of code that performs a specific task. Functions have parentheses `()` at the end, for example `date()` is a function which returns the current date and time.

An **argument** is a piece of information that you pass into a function, which will change the way the function behaves. Let's look at an example.

The code below will use the function called `round()`. Within this function (inside the parentheses) I can enter an **argument**. I've entered the mathematical constant pi here as the argument.

The function `round()` with pi entered as an argument (inside the parentheses) will just round the value to the nearest whole number. Try this yourself in RStudio now.

```
round(3.14159265359)
#> [1] 3
round(pi) # or you can just type `pi`
#> [1] 3
```

However, what if I wanted this output to two-decimal places? No problem! Simply add another argument. Some functions will contain multiple arguments, but for now let's keep it simple.

Run the code below and check your console to see the output.

```
round(3.14159265359, digits = 2)
```

Here we have use a `,` comma to separate the arguments within the parentheses. The argument `digits = 2` can be used to change the behaviour of this function by rounding pi to two decimal places. An **argument** will give a function some more specific guidance.

As you advance further with R code you'll be using lots of different functions and arguments.

If at any point you want to learn about a function, we can use `?`. Try running the code below:

```
?round()
```

If you ran it successfully you will notice in the **output panel** (bottom right) that the **help** tab is now showing documentation to explain how this function works, including all the possible arguments.

You can use `?` at any time to see how a function works.

2.3 Comments = Human Notes

You might be a little worried that eventually you are going to have to remember so much code. The base version of R contains thousands of functions, even before we start using **packages**! (more on packages later). One concern people have with code is needing to remember everything.

To deal with this issue, when writing our code, we should use comments in our R script. A **comment** is a piece of text which you can add next to your code as a note. I like to think of comments as "human notes", we can read them but RStudio will ignore them.

In order to tell RStudio where our human notes begin we need to use `#` hashtag. Anything written after the hashtag is not processed or read by RStudio. Check out the example below:

```r
date() #this will ask for today's date

round(3.14159265359, digits = 2) #this rounds a value to two decimal places
```

Rstudio will run those **functions** but the notes are there for your benefit. You should use notes routinely when writing code for two important reasons:

1. It will help remind you how your code works if you look at it again in the future.
2. If you send or share your code with someone else, your notes can help them to understand what you've done and why.

Therefore, here we have our first important rule:

Always make use of comments when coding in RStudio (#)

Before moving on to the next section, head back into RStudio and practice adding some notes to your R script. Remember to use the `#` hashtag. Once you've finished writing your notes, hit enter and then the script moves to the next line ready for the next bit of code.

2.4 Creating Objects

Another important feature is knowing how to create an **object**. Variables and/or data can be assigned to objects, however, you need to create the object yourself. Let's look at an example. Run the code below:

```r
myobject <- c(1, 2, 3, 4)
```

Let's talk through this code step by step:

- What you might notice is that I have created an object called `myobject` (a truly inspirational name). This is on the very left of this line of code. You can call an object anything you like, so feel free to call it something else.
- Next I have used an arrow `<-`. This means that anything written on the right hand side of this arrow will be committed to this object. This arrow is called an assignment operator.
- Then I used `c()` which is a function that allows us to list values. I created a mini data set with the numbers 1-4.

Go ahead if you haven't already and run that bit of code, nothing particularly thrilling will happen and you may notice that the console does not return anything exciting. However, take a look at the **environment** panel (top right) and what do you notice?

If it has worked, you should spot that there is an object called `myobject` (or whatever you called yours) as shown in Figure 2.1.

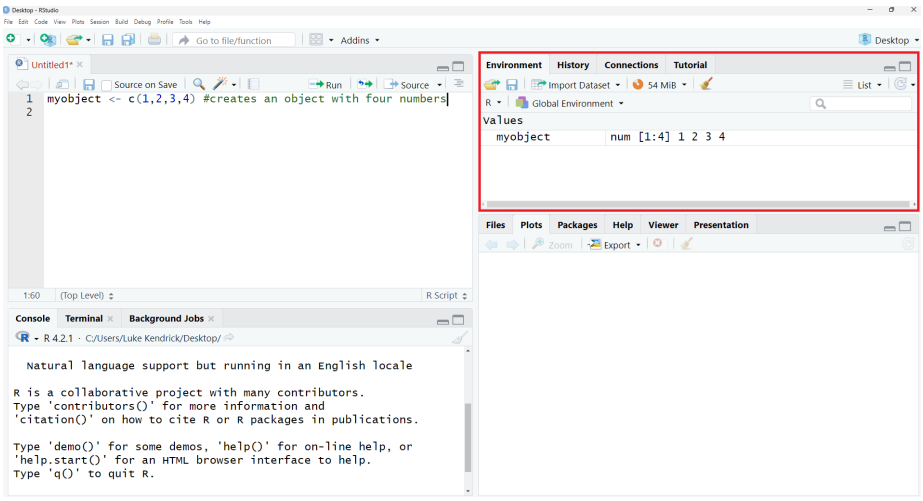

Figure 2.1: The environment contains the new object called 'myobject'.

In the R world:

- Everything that exists is an object.
- Everything that happens is a function.

Almost anything can be committed to an object in RStudio. For example, run and adapt the code below. Change "Luke" to your own name! Then check that your environment looks like Figure 2.2.

```
name <- "Luke"
answer <- 12345*54321
pi <- round(3.14159265359, digits = 2)
```

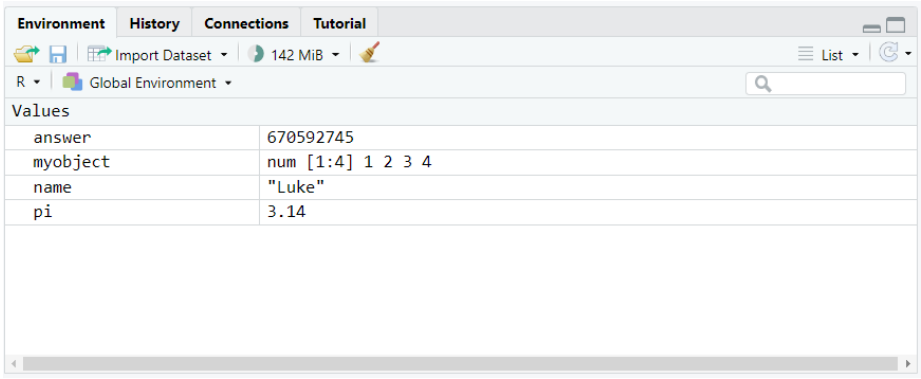

Figure 2.2: The environment contains multiple new objects.

We can also print an object to the console. Essentially this will ask RStudio to just show the contents of an object in the console. You should try this with the `print()` function, see below.

```
print(myobject)
print(name)
print(answer)
print(pi)
```

2.4.1 Naming and Removing Objects

I did say that you can call an object anything you like. One thing that will help is to try and keep object names short. That is because often you might need to keep typing out object names, particularly if you use the object frequently (e.g., an object that contains your data set for analysis).

Generally, avoid using multiple words and keep it short and informative.

```
what_is_my_first_name <- "Luke" # this is not a good object name. it is too long.
name <- "Luke" # this is a short informative object name
```

Finally, if you want to remove an object from your environment you can use the code below:

```
rm(myobject, name, answer, pi) # will remove these four objects.

rm(list = ls()) # will remove EVERYTHING from the environment.
```

2.5 Troubleshooting and Getting Help with R

During the early stages of learning to code it is not uncommon to come across error messages. Sometimes the code will not work and this can be frustrating. Also, some error messages are a little tricky to interpret. Thankfully there are options for support to troubleshoot issues.

Firstly, remember to use `?` in front of a function to check out the documentation to ensure you are using it correctly. If I wanted to learn about the `rm()` function from the previous section I would use:

```
?rm()
```

Secondly, if you come across an error message and you cannot figure it out/it looks a load of gibberish, copy and paste the error message into Google. It is likely that someone else would have experienced this error and there is probably a solution already online.

Finally, one of the most common issues when learning to code is the presence of a typo. When you code in RStudio it needs to be written very specifically and how it expects it to be written. RStudio will not work if you make a typo. Similarly, R code is case sensitive. `myobject` is not the same as `MYOBJECT` or `mYobject`.

Take your time to check you code carefully when you are starting out in R, most errors tend to be small typos.

Therefore, here we have our second important rule:

Check your code for typos and enter it exactly as R expects

2.6 Chapter Summary

You should now have an understanding of how functions and arguments work. You should also have ran your very first lines of code! Congratulations, you are officially an R coder. You should also now be able to create objects, name them appropriately, and remove them from the environment. Finally, you should be aware of two key rules: (1) use comments to make notes with you code and (2) enter code exactly intended and check out for typos. In the next chapter you will begin to work with packages and learn how to import a data set into RStudio ready for data analysis.

Chapter 3

Packages and Importing Data

In this chapter we will cover two important features of RStudio. Firstly is the use of packages. Packages allow you to access a variety of different functions. Additionally, before you begin working with any data you will need to somehow actually get that data into RStudio via a working directory.

3.1 What are Packages?

When you first open RStudio, you will have access to many base R functions. In fact there are a great deal of functions that exist in the base version of R. In fact, you have already used some in the previous chapter, for example, `date()` or `round()` are considered base R functions.

Base R is essentially the "basic subscription" version of RStudio without any additional apps installed.

Often we might need some quite specialised functions. In order to access these we need to install and load `packages`. You can think of `packages` as a bit like your mobile phone apps...in that you need to download them, install them, and then open them to access their specific functions.

One **`key`** package that is required for learning and using R for Psychology at Royal Holloway is the `tidyverse` package. `Tidyverse` is essentially a collection of packages, see more here.

3.1.1 Installing a Package

Tidyverse is already installed on most Royal Holloway campus computers and so this step is not necessary. But still worth a read as in future you may need to install other packages beyond the tidyverse.

To install tidyverse we can use the `install.packages()` function. See below:

```
install.packages("tidyverse") # install the tidyverse package
```

Note you'll see I also followed my own advice and used human notes with the hashtag `#`.

You'll notice that the name of the package needs to be in quotation marks. When you run this line of code, RStudio will get to work installing the package. It might take a couple of minutes and you will see this happening in the console (bottom left panel).

3.1.2 Loading a Package

Just because you have installed a package does not yet mean you are ready to use it. Next you need to load it. To do this you can use the `library()` function below:

```
library(tidyverse) # load the tidyverse package
```

Note that we do not need to use quote marks.

Remember, you only need to install a package once on your computer. However, each time you reopen RStudio (whether at home or on a campus PC), you'll need to load the necessary libraries using the `library()` function.

3.1.3 Checking Packages

You can check which packages you have installed and loaded at anytime. Head to the "packages" tab on the bottom right panel. If the package is listed then it is installed. If there is a tick in the box on the left, that also means the package has been loaded and is ready to go! Why not try and un-tick and then tick tidyverse in this panel and see what happens in the console. See Figure 3.1 below.

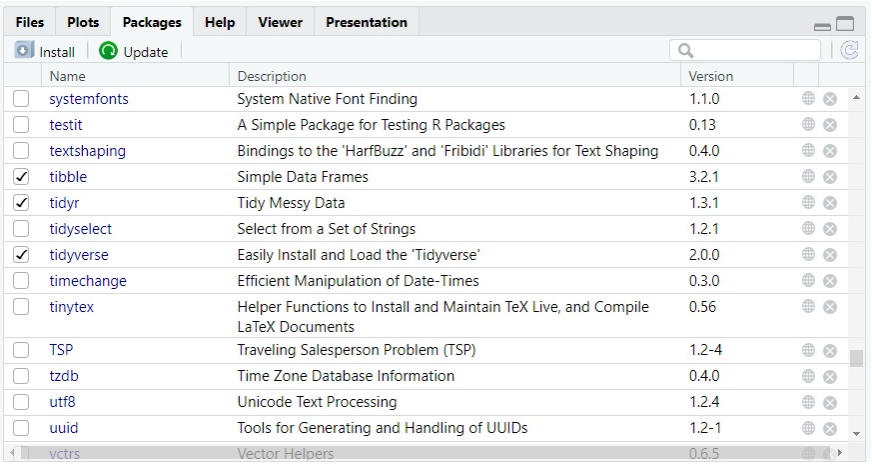

Figure 3.1: An example of the packages tab with the Tidyverse package loaded.

3.2 Importing Data

Now we have Tidyverse loaded, you can begin thinking about importing some data. There are different way to import data, and you might actually notice an `Import Dataset` in the environment panel. However, the best way to import data is via a `Working Directory` and using a Tidyverse function called `read_csv`. Let's go through this bit by bit.

3.2.1 Set the Working Directory

The working directory is a folder on your computer that RStudio will look in to find any data files. It is also where RStudio can save any output files too.

Before setting your working directory it is important to think about your file structure on your computer. For example, you might consider creating a special folder to save any of your RStudio work. You can then link this special folder to RStudio using the steps below.

Head to the `Session` tab at the top of the RStudio window and then select `Set Working Directory` -> `Choose Directory`. See Figure 3.2 below.

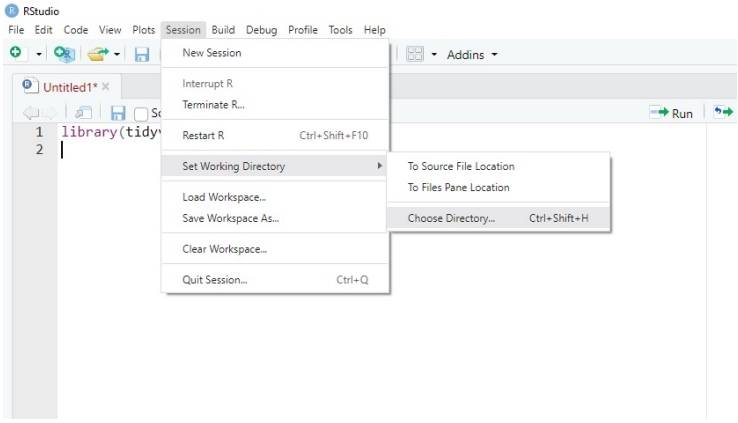

Figure 3.2: Set the working directory

This will open up a file browser on your computer. Select the folder on your computer that you have designated as your `Working Directory`. The one in Figure 3.3 is called `R_Folder` but you might call yours something else if you like, for example, `StartR`. You'll notice that the folder I have selected below is empty because it is a new folder with nothing in it...yet!

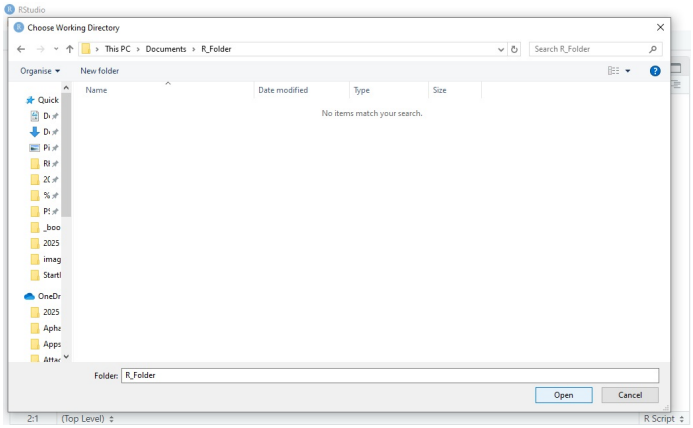

Figure 3.3: Head into the folder that you will use as your working directory in this session

In case you forget, you can check your working directory at any point using this line of code:

```
getwd() # will check the current working directory.
```

You should see in the console (bottom left) that RStudio will tell you the location of the folder on your computer that is now your designated working directory.

3.2.2 Importing Data

RStudio primarily likes to work with a file format called `.csv`. This stands for comma separated values. There are ways to import other types of files (e.g., SPSS, excel, text) but during this course we will only work with `.csv` files.

Let's try and get a basic data set in to RStudio. Download the data set below and *remember* save it into your designated working directory folder. For reference, the data are also presented below.

Download the data set

Table 3.1: A table of ten people containing their age (in years) and height (in cm).

id	age	height
1	22	167
2	26	178
3	18	173
4	32	169
5	19	183
6	39	172
7	41	159
8	46	164
9	32	153
10	25	159

3.2.3 read_csv()

Once the data file is in your working directory folder we can try to import it using `read_csv()`.

```
data <- read_csv("height.csv") # import the height.csv data set and save it as an obje
```

This line of code here begins to combine everything we have covered in this StartR short course so far. Take a moment ro work through the explanations below to ensure you understand:

- `data` is an **object**. I have called it data as it is just a short 4 letter word and is our data set.

- `<-` is the **assignment operator**. It will assign things to the right side to our object on the left side called `data.`
- `read_csv()` is a **function** which will read data from a .csv file.
- `"height.csv"` is an **argument** which tells the function how to behave. In this case it will look for a specific file called height.csv.
- It will look for this file in the `working directory`.
- If this works successfully you should see an object called `data` in the **environment** panel (top right), see Figure 3.4.
- Notice that "height.csv" is spelled exactly to match the file name. No capital letters or spaces, which is how it appeared when you downloaded it.

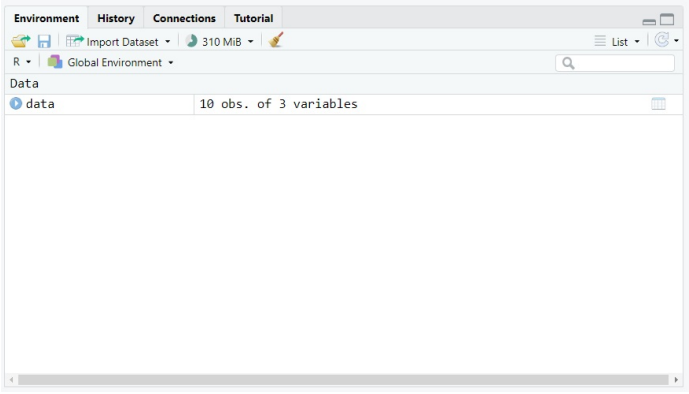

Figure 3.4: The environment panel with the newly created 'data' object

You might also notice there is some very basic information visible to the right of the object in the environment. For example, there are `10 obs. of 3 variables`. This tells us there are 10 observations across three different variables.

Finally, if you double click on the object name `data`, it will open the data file in a new tab.

You can also achieve the same result using:

```
view(data) # will open the data in a new tab
```

A final note on `read_csv()`.

You might learn that there is a similar function called `read.csv` (with a dot `.` not an underscore `_`). Do not use read.csv. Always use the version with an underscore, which comes from the tidyverse package.

3.2.4 Some (very) Basic Descriptive Statistics

In the final part of this chapter, you will calculate some very basic descriptive statistics. I wanted to minimise any "stats" stuff in this short course as the aim is really to learn about the RStudio layout and how it works before we properly work with data.

However, the code below will allow you to practice a little more understanding R functions, running code, and reading output.

We need a new package for this final section:

```
install.packages("psych") # install Psych package
library(psych) # load the psych package
```

How about you work out some basic descriptive statistics for this data file. Using the `psych` package, you could use:

```
describe(data) # produce descriptive statistics
```

Run that now and check the output. The first line you might notice that id (identification number) has a mean of 5.50. This is meaningless and actually we could remove the ID column. How could we do this?

3.2.5 Using the Pipe Operator

Do you remember right at the start of chapter one I used the example below which is a three step command:

```
`stand up` and then,
`walk towards the window` and then,
`return to your seat`
```

With our data we actually want to do the following which is also a three step command:

```
use `data` and then,
`remove id column` and then,
`describe` it.
```

We can use something called a pipe operator `%>%` which essentially means "and then". What will happen next is we will write out a set of instructions, line by line. Take a look at the code below and then run it:

```
data %>%
  select(-id) %>%
  describe()
```

The three lines do the following:

1. Take our object `data` …and then…
2. Use `select()` to remove id. We do this adding a minus sign `-` before id…and then
3. `describe()` the results.

Hopefully you can now see in the console the new output with the id column removed.

You will also see that the mean age was 30 years (SD = 9.64) and the mean height was 167.70 centimeters (SD = 9.25).

Congratulations! You have successfully produced some descriptive statistics.

3.3 Chapter Summary

In this chapter you have learnt about what packages can do for us. You might come across several different packages in the future and so you now know how to install packages and then load them. Another important task is being able to set the working directory. This is the folder on your computer that RStudio will use to find any data files. Next, you would have successfully imported a data file using read_csv(). Finally, you should have use code to produce some very simple descriptive statistics.