Suffolk 2019

Contents

About this course

This course is an introduction R, RStudio and statistics. Our primary aims are to get comfortable working with R and to be able to prepare, manipulate, analyse and visualise data..

Agenda

1 Introduction to R and RStudio

1.1 Learning objectives

In this session, we will have a look at R and RStudio. We will interact with both and use the various components of RStudio.

1.1.1 What is R?

R is an environment for statistical computing and graphics. RStudio is an editor or integrated development environment (IDE) that makes working with R much more comfortable.

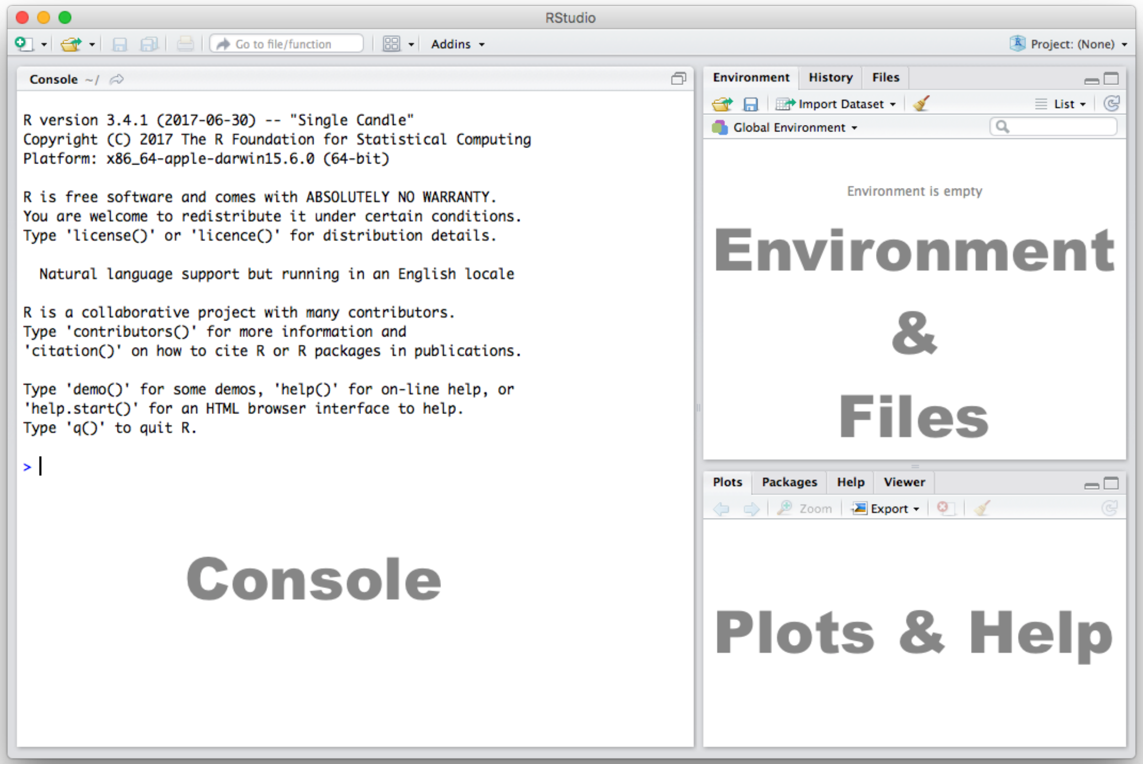
To install R and RStudio on your computer, download both from the following sources:

- Download R from The Comprehensive R Archive Network (CRAN)
- Download RStudio from RStudio.com

Keep both R and RStudio up to date. That means go online and check for newer versions. In case there are new versions, download those and re-install.

1.1.2 RStudio

Let's get acquainted with R. When you start RStudio for the first time, you'll see three panes:



```
R version 3.4.1 (2017-06-30) -- "Single Candle"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)

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.

>
```

1.1.3 Console

The Console in RStudio is the simplest way to interact with R. You can type some code at the Console and when you press ENTER, R will run that code. Depending on what you type, you may see some output in the Console or if you make a mistake, you may get a warning or an error message.

Let's familiarize ourselves with the console by using R as a simple calculator:

```
2 + 4
```

```
[1] 6
```

Now that we know how to use the + sign for addition, let's try some other mathematical operations such as subtraction (-), multiplication (*), and division (/).

```
10 - 4
```

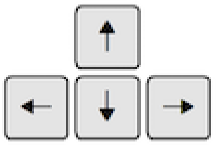
```
[1] 6
```

```
5 * 3
```

```
[1] 15
```

```
7 / 2
```

```
[1] 3.5
```



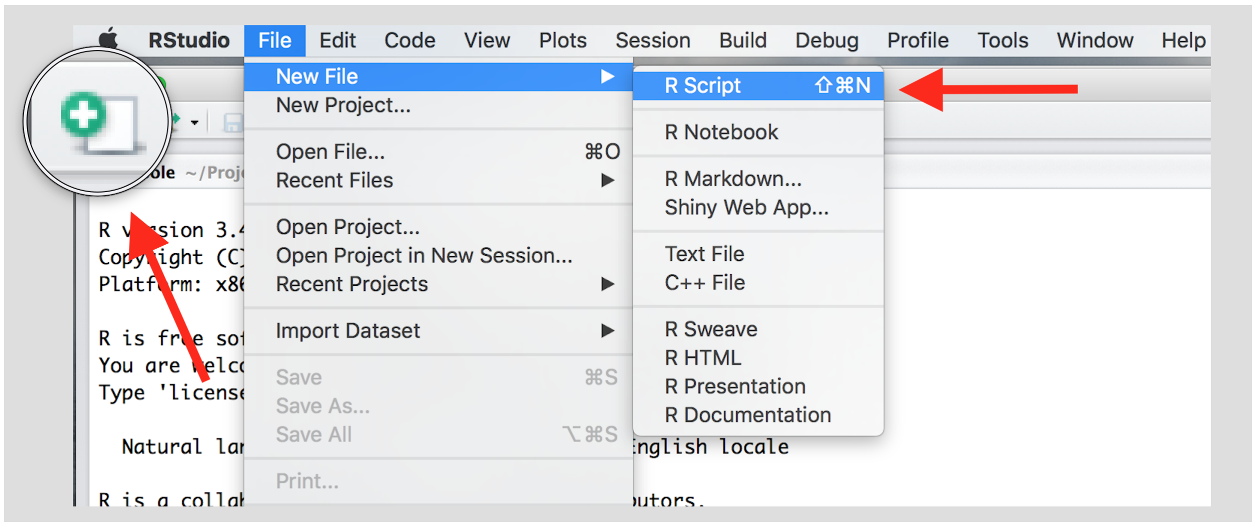
You can use the cursor or arrow keys on your keyboard to edit your code at the console:- Use the UP and DOWN keys to re-run something without typing it again- Use the LEFT and RIGHT keys to edit

Take a few minutes to play around at the console and try different things out. Don't worry if you make a mistake, you can't break anything easily!

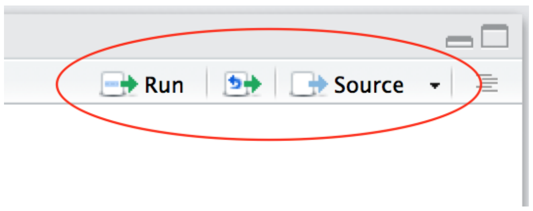
1.1.4 Scripts

The Console is great for simple tasks but if you're working on a project you would mostly likely want to save your work in some sort of a document or a file. Scripts in R are just plain text files that contain R code. You can edit a script just like you would edit a file in any word processing or note-taking application.

Create a new script using the menu or the toolbar button as shown below.



Once you've created a script, it is generally a good idea to give it a meaningful name and save it immediately. For our first session save your script as **seminar1.R**



Familiarize yourself with the script window in RStudio, and especially the two buttons labeled **Run** and **Source**

There are a few different ways to run your code from a script.

One line at a time	Place the cursor on the line you want to run and hit CTRL-ENTER or use the **Run** button

Multiple lines	Select the lines you want to run and hit CTRL-ENTER or use the **Run** button
Entire script	Use the **Source** button

2 R-syntax, data structures and types

2.1 Seminar

In this session we introduce R-syntax, and data types.

2.1.1 Functions

Functions are a set of instructions that carry out a specific task. Functions often require some input and generate some output. For example, instead of using the `+` operator for addition, we can use the `sum` function to add two or more numbers.

```
sum(1, 4, 10)
```

```
[1] 15
```

In the example above, `1, 4, 10` are the inputs and 15 is the output. A function always requires the use of parenthesis or round brackets `()`. Inputs to the function are called **arguments** and go inside the brackets. The output of a function is displayed on the screen but we can also have the option of saving the result of the output. More on this later.

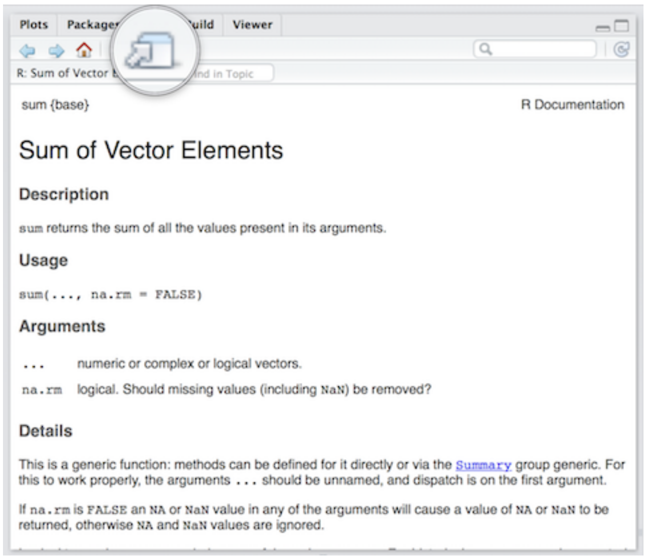
2.1.2 Getting Help

Another useful function in R is `help` which we can use to display online documentation. For example, if we wanted to know how to use the `sum` function, we could type `help(sum)` and look at the online documentation.

```
help(sum)
```

The question mark `?` can also be used as a shortcut to access online help.

```
?sum
```



sum {base} R Documentation

Sum of Vector Elements

Description

`sum` returns the sum of all the values present in its arguments.

Usage

```
sum(..., na.rm = FALSE)
```

Arguments

`...`	numeric or complex or logical vectors.
`na.rm`	logical. Should missing values (including `NaN`) be removed?

Details

This is a generic function: methods can be defined for it directly or via the `Summary` group generic. For this to work properly, the arguments `...` should be unnamed, and dispatch is on the first argument.

If `na.rm` is `FALSE` an `NA` or `NaN` value in any of the arguments will cause a value of `NA` or `NaN` to be returned, otherwise `NA` and `NaN` values are ignored.

Use the toolbar button shown in the picture above to expand and display the help in a new window.

Help pages for functions in R follow a consistent layout generally include these sections:

Description	A brief description of the function
Usage	The complete syntax or grammar including all arguments (inputs)
Arguments	Explanation of each argument
Details	Any relevant details about the function and its arguments
Value	The output value of the function
Examples	Example of how to use the function

2.1.3 The Assignment Operator

Now we know how to provide inputs to a function using parenthesis or round brackets `()`, but what about the output of a function?

We use the assignment operator `<-` for creating or updating objects. If we wanted to save the result of adding `sum(1, 4, 10)`, we would do the following:

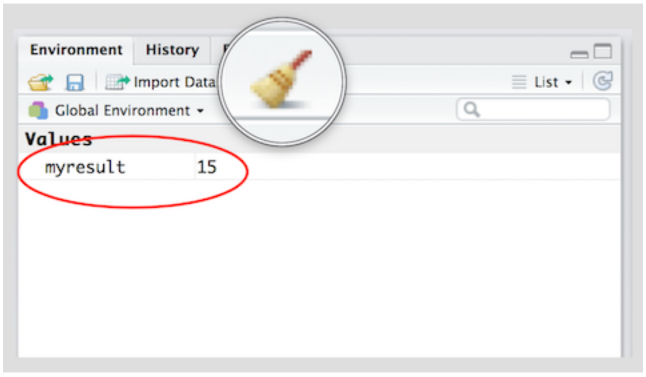
```
myresult <- sum(1, 4, 10)
```

The line above creates a new object called `myresult` in our environment and saves the result of the `sum(1, 4, 10)` in it. To see what's in `myresult`, just type it at the console:

```
myresult
```

```
[1] 15
```

Take a look at the **Environment** pane in RStudio and you'll see `myresult` there.



```
myresult    15
```

To delete all objects from the environment, you can use the **broom** button as shown in the picture above.

We called our object `myresult` but we can call it anything as long as we follow a few simple rules. Object names can contain upper or lower case letters (`A-Z`, `a-z`), numbers (`0-9`), underscores (`_`) or a dot (`.`) but all object names must start with a letter. Choose names that are descriptive and easy to type.

Good Object Names	Bad Object Names
result	a
myresult	x1
my.result	this.name.is.just.too.long
my_result	
data1	

2.1.4 Sequences

We often need to create sequences when manipulating data. For instance, you might want to perform an operation on the first 10 rows of a dataset so we need a way to select the range we're interested in.

There are two ways to create a sequence. Let's try to create a sequence of numbers from 1 to 10 using the two methods:

1. Using the colon `:` operator. If you're familiar with spreadsheets then you might've already used `:` to select cells, for example `A1:A20`. In R, you can use the `:` to create a sequence in a similar fashion:

```
1:10
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

1. Using the `seq` function we get the exact same result:

```
seq(from = 1, to = 10)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

The `seq` function has a number of options which control how the sequence is generated. For example to create a sequence from 0 to 100 in increments of 5, we can use the optional `by` argument. Notice how we wrote `by = 5` as the third argument. It is a common practice to specify the name of argument when the argument is optional. The arguments `from` and `to` are not optional, se we can write `seq(0, 100, by = 5)` instead of `seq(from = 0, to = 100, by = 5)`. Both, are valid ways of achieving the same outcome. You can code whichever way you like. We recommend to write code such that you make it easy for your future self and others to read and understand the code.

```
seq(from = 0, to = 100, by = 5)
```

```
 [1]   0   5  10  15  20  25  30  35  40  45  50  55  60  65  70  75  80
[18]  85  90  95 100
```

Another common use of the `seq` function is to create a sequence of a specific length. Here, we create a sequence from 0 to 100 with length 9, i.e., the result is a vector with 9 elements.

```
seq(from = 0, to = 100, length.out =  9)
```

```
[1]   0.0  12.5  25.0  37.5  50.0  62.5  75.0  87.5 100.0
```

Now it's your turn:

- Create a sequence of **odd** numbers between 0 and 100 and save it in an object called `odd_numbers`

```
odd_numbers <- seq(1, 100, 2)
```

- Next, display `odd_numbers` on the console to verify that you did it correctly

```
odd_numbers
```

```
 [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45
[24] 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91
[47] 93 95 97 99
```

- What do the numbers in square brackets `[ ]` mean? Look at the number of values displayed in each line to find out the answer.
- Use the `length` function to find out how many values are in the object `odd_numbers`.
  - HINT: Try `help(length)` and look at the examples section at the end of the help screen.

```
length(odd_numbers)
```

```
[1] 50
```