Introduction to Statistics

Contents

About this course

This course is an introduction to statistics, R and RStudio. Our primary aims are to introduce you to and help you become familiar with RStudio and quantitative methodologies critical to your development as an analyst.

By the end of the course, you should be able to understand fundamental research methods, apply them to real world problems and acquire competency in performing statistical functions using R.

Slides day 1

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.

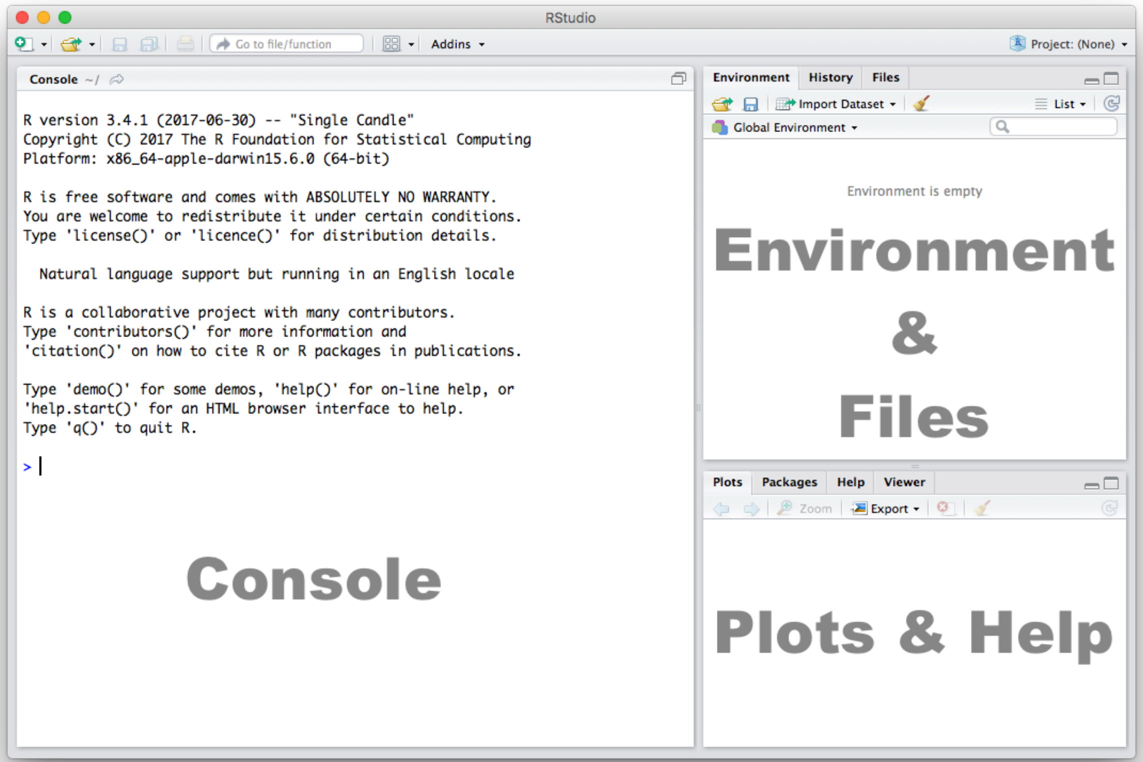
To install R and RStudio on your computer, download both from the following sources:

- 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
```

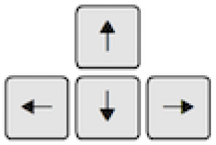
```
5 * 3
```

```
[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

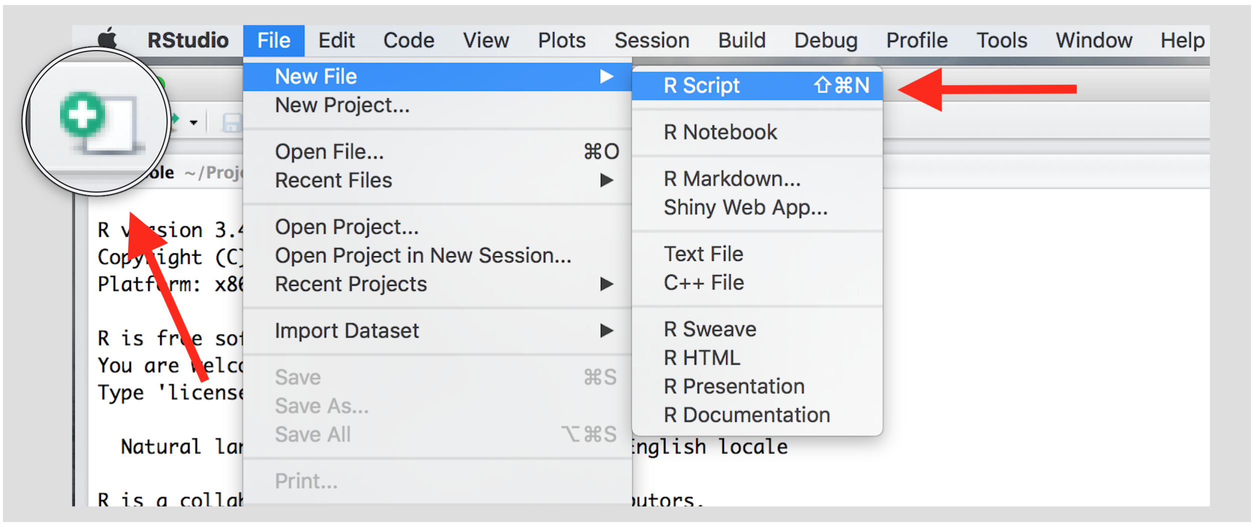


Take a few minutes to play around at the console and try different things out. Don't worry if you make a mistake, you can't break anything easily!

1.1.4 Scripts

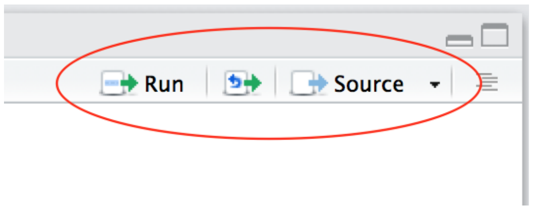
The Console is great for simple tasks but if you're working on a project you would mostly likely want to save your work in some sort of a document or a file. Scripts in R are just plain text files that contain R code. You can edit a script just like you would edit a file in any word processing or note-taking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

2.1 Seminar

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

2.1.1 central t

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.

```r
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.

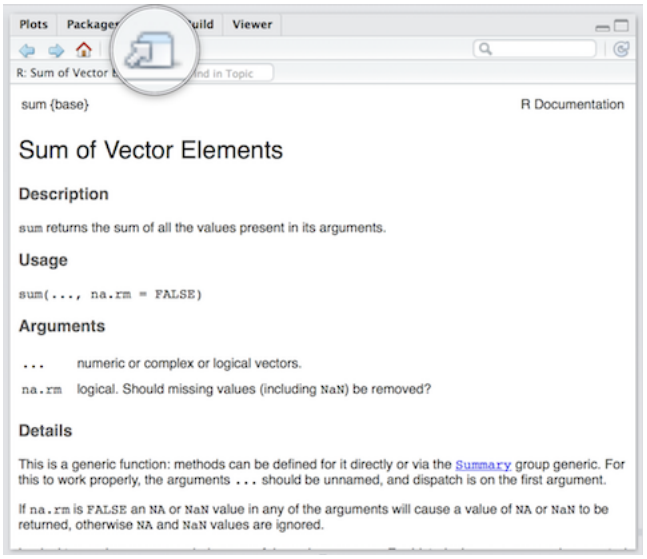
2.1.2 Getting Help

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.

```r
help(sum)
```

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

```r
?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:

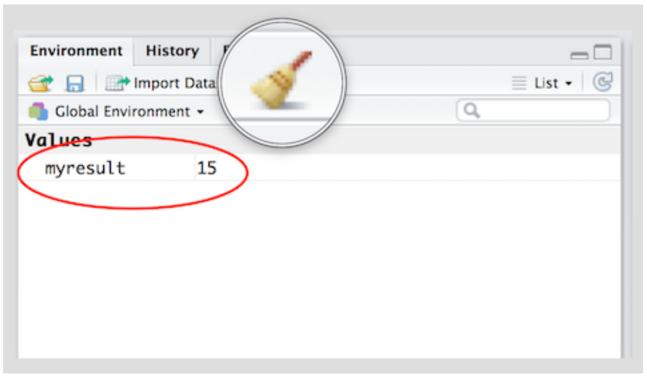
```
myresult <- sum(1, 4, 10)
```

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 Vectors and subsetting

A vector is one dimensional. It can contain one element in which case it is also called a scalar or many elements. We can add and multiply vectors. Think of a vector as a row or column in your excel spreadsheet.

To create a vector, we use the `c()` function, where c stands for collect. We start by creating a numeric vector.

```r
vec1 <- c(10, 47, 99, 34, 21)
```

Creating a character vector works in the same way. We need to use quotation marks to indicate that the data type is textual data.

```r
vec2 <- c("Emilia", "Martin", "Agatha", "James", "Luke", "Jacques")
```

Let's see how many elements our vector contains using the `length()` function.

```r
length(vec1)
```

```
[1] 5
```

```r
length(vec2)
```

```
[1] 6
```

We need one coordinate to identify a unique element in a vector. For instance, we may be interested in the first element of the vector only. We use square brackets `[]` to access a specific element. The number in square brackets is the vector element that we wish to see.

```r
vec1[1]
```

```
[1] 10
```

To access all elements except the first element, we use the `-` operator

```r
vec1[-1]
```

```
[1] 47 99 34 21
```

We can access elements 2 to 4 by using the colon `:` operator.

```r
vec1[2:4]
```

```
[1] 47 99 34
```

We can access non-adjacent elements bu using the collect function `c()`.

```r
vec1[c(2,5)]
```

```
[1] 47 21
```

Finally, we combine the `length()` function with the square brackets to access the last element in our vector.

```r
vec1[ length(vec1) ]
```

```
[1] 21
```

2.1.5 Matrices

A matrix has two dimensions and stores data of the same type, e.g. numbers or text but never both. A matrix is always rectangular. Think of it as your excel spreadsheet - essentially, it is a data table.

We create a matrix using the `matrix()` function. We need to provide the following arguments:

```r
mat1 <- matrix(
  data = c(99, 17, 19, 49, 88, 54),
  nrow = 2,
  ncol = 3,
  byrow = TRUE
)
```

Argument	Description
data	the data in the matrix
nrow	number of rows
ncol	number of columns
byrow	TRUE = matrix is filled rowwise

To display the matrix, we simply call the object by its name (in this case mat1).

```
mat1
```

```
     [,1] [,2] [,3]
[1,]   99   17   19
[2,]   49   88   54
```

To access a unique element in a matrix, we need 2 coordinates. First, a row coordinate and second, a column coordinate. We use square brackets and separate the coordinates with a comma `[ , ]`. The row coordinate goes before the comma and the column coordinate after.

We can access the the second row and third column like so:

```
mat1[2, 3]
```

```
[1] 54
```

To display an entire column, we specify the column we want to display and leave the row coordinate empty like so:

```
# display the 2nd column
mat1[ , 2]
```

```
[1] 17 88
```

Similarly, to display the entire second row, we specify the row coordinate but leave the column coordinate empty.

```
mat1[2, ]
```

```
[1] 49 88 54
```

2.1.6 Arrays

Arrays are similar to matrices but can contain more dimensions. You can think of an array as stacking multiple matrices. Generally, we refer to the rows, columns and layers in array. Let's create an array with 2 rows, 3 columns and 4 layers using the `array()` function.

```
arr1 <- array(
  data = c(1:24),
  dim = c(2, 3, 4)
)
```

To display the object, we call it by its name.

```
arr1
```

```
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

, , 3

     [,1] [,2] [,3]
[1,]   13   15   17
[2,]   14   16   18

, , 4

     [,1] [,2] [,3]
[1,]   19   21   23
[2,]   20   22   24
```

We can subset an array using the square brackets `[]`. To access a single element we need as many coordinates as our object has dimensions. Let's check the number of dimensions in our object first.

```r
dim(arr1)
```

```
[1] 2 3 4
```

The `dim()` function informs us that we have 3 dimensions. The first is of length 2, the second of length 3 and the fourth of length 4.

Access the second column of the third layer on your own.

```r
arr1[ , 2, 3]
```

```
[1] 15 16
```