Introduction to R: Session 1

15 October 2025

Table of Contents

Original material developed by Alexis Robert and Sophie Meakin.

Updated (2025) by Alexis Robert.

Introduction

Welcome to *Introduction to R*!

Contents

This session introduces the basics of R.

Notation

This exercise will contain examples of code and its output, as well as instructions to edit or write code yourself. We've used the following symbols to highlight certain instructions:

- ⭐ A key piece of information e.g. a useful R function.
- 💻 An instruction to copy-and-run code from the exercise, or write your own code, in your R Studio session.
- ❓ A question, related to the material or code output.

Resources

- The Epidemiologist R Handbook - a comprehensive reference guide to using R for applied epidemiology and public health.
- R for Data Science - a general guide to using R for data science.
- Stack Overflow - general R questions answered by the community.

R basics

One of the most important uses of any statistical software, including R, is to analyse data that you may have collected yourself or that have been given to you. You will learn a lot more about data analysis during your MSc. In this section, We will introduce how to use R to access data that you already have on your computer, and how to store the results of your analysis or save an updated version of the data, on your computer.

R Studio

Your R studio session will have four main panels:

1. **Script** (top-left) - this is where you can write, save and run longer bits of code.
2. **Environment** (top-right) - this is where you can see objects that you have created in your current R session.
3. **Console** (bottom-left) - this is where you can write and run very short bits of code.
4. **Files, plots, etc** (bottom-right) - this is where you can see your files, or where plots will appear.

Basic R expressions

This section introduces the commands to do simple operations in R, such as arithmetic, or how to make objects.

Arithmetic operators

These operators are used to carry out mathematical operations, like addition or multiplication:

- +: addition
- -: subtraction
- *: multiplication
- /: division
- ^: exponent

Examples:

```r
46 + 3
13 / 8
2^5
```

💻 Try using some of these operators in your R Studio console (the bottom-left panel).

The assignment operator

⭐ The assignment operator in R is `<-` and is used to assign values to named objects.

⭐ If you run every line in the console (bottom left panel), you will not easily see the previous lines of code, that's why we use the script (top left panel).

- To execute a line of code from your script: place your cursor on the line you want to run and click on `Run` (top right of the script panel) OR place your cursor and use the shortcut `CTRL+ENTER` (`CMD+ENTER` on Mac).
- To run multiple lines of code, select them with your mouse, and use `CTRL+ENTER` (or click on `Run`).
- To run the whole script, use `Source` (top right of the script panel), or use the shortcut `CTRL+SHIFT+ENTER` (`CMD+SHIFT+ENTER` on Mac)

```r
x <- 10       # The object called x is assigned the value 10
x             # Print the object x

y <- 10 + 20  # The object called y is assigned the value 10 + 20
y             # Print the object y

z <- "hello"  # The object called z is assigned the text "hello"
z             # Print the object z
```

💻 Look at your Environment panel (top-right) of your R Studio session, then run the above code in your console. Look in the environment panel again: what has changed?

Note that the names of objects are case sensitive (i.e. `Y` and `y` are distinct names) and must not start with numbers or contain spaces. The best approach is to use meaningful object names and stick to using all lower case; if you need to use spaces, then you can use an underscore (_).

- Examples of good object names: `raw_data`, `clean_data`, `country_names`.
- Examples of bad object names: `raw data`, `CLEAN_data`, `dasyg`, `1x`.

❓ Discuss *why* each of the examples of bad object names are bad.

💻 Try making some of your own objects, e.g. assign the value `"2025"` to the object `year`.

Relational operators

Relational operators are used to compare different values:

- < less than
- > greater than
- <= less than or equal to
- >= greater than or equal to
- == equal to
- != not equal to

Expressions using relational operators will return `TRUE` or `FALSE`.

💻 Copy the below code into the console of your R Studio terminal.

```
x <- 3      # Assign the value 3 to the object called x
y <- 11     # Assign the value 11 to the object called y
```

Examples of using relational operators:

```
# Test: is x less than 4?
x < 4

# Test: is y greater than or equal to x
y >= x

# Test: is x equal to 5?
x == 5

# Test: is y not equal to 22
y != 22
```

💻 Try using some of these operators in your R Studio console, e.g. to test if `2^11` is bigger than `3*729`.

The `c()` function

The `c()` function is used to combine multiple values into a *vector* or a *list*. For example, multiple numeric values can be combined into a vector:

```
c(3, 1, 10.5)
```

Multiple text strings can also be combined:

```
c("Welcome", "to", "R", "!")
```

A vector can be assigned to an object, elements of the vector are accessed with brackets `[]`:

```
a <- c(5, 4, 10)
a[1]
a[2] + 3
```

❓ What is the value of the third element of a?

💻 Try running `a + 3`, what results do you get?

If both numbers and text are combined with the `c()` function, then all the numeric values are transformed into text:

```
b <- c("LSHTM", 4)
b
```

❓ When running `b`, how do you see that the second element was transformed into text?

❓ What happens if you try adding 2 to the second element of b? Why?

Inspecting data frames

Datasets are usually stored as data frames, which are objects that can contain several columns (i.e. features) of different types. In this section, we will import a dataframe, and introduce basic functions to inspect the data contained in the dataset.

There are several ways to import data into R, in this session we will use a dataset included in a package. Packages are "extensions" of R, often developed by the community. They bring additional functions and datasets not included in the base version of R. We will give you more information on what packages are, and explore how to import datasets from online or local files in future sessions.

For this example, we will use a dataset included in the package `palmerpenguins`. This dataset contains data on hundreds of penguins, collected over several expeditions. First, we use the function `install.packages` to install this dataset onto our computer.

```
install.packages("palmerpenguins")
```

We then use `library` to import the content of `palmerpenguins` in the R session.

```
library(palmerpenguins)
```

`palmerpenguins` contains two objects, for now we will focus on `penguins_raw`, a dataframe containing penguin size data and observations for 344 penguins. We want to only focus on a few columns of `penguins_raw` for this exercise, so we're going to select these columns, and get rid of the others. You can check the column names of a dataframe by using the function `colnames()`:

```
colnames(penguins_raw)
```

We only want to select the id of the penguin, where they were located, the length of their flipper, their weight and sex. Therefore, we create a new object, `data_penguins`, which contains the columns of interest. We use the brackets to indicate the columns we are interested in.

```
data_penguins <- penguins_raw[, c("Individual ID", "Region", "Flipper Length (mm)", "Body Mass (g)", "Sex")]
```

`data_penguins` contains all the rows included in penguins_raw, but only the column Individual ID, Region, Flipper Length, Body Mass, and Sex. As previously discussed, we should avoid used upper case characters, spaces, brackets, and parentheses when naming objects, since that leads to errors when we try to access these objects. This is also true for column names, so let's rename the columns of `data_penguins`.

```
colnames(data_penguins) <- c("id", "region", "length", "weight", "sex")
```

Now we have imported our data, we can look at it in one of two ways:

1. 💻 Click on the `data_penguins` object in the 'Environment' panel (top-right). A new window will open in front of your script pane showing the data. Have a look!
2. 💻 Run the command `data_penguins` in your Rstudio console. The data will print in the console. Alternatively, the command `head(data_penguins)` will print the top 6 lines only:

```
head(data_penguins)
```

Notice that there are `NA` values in the `length`, and `sex` columns. These indicate missing values, where no data is available.

You can also get the number of columns and number of rows in the data set with `ncol()` and `nrow()`, respectively:

```
ncol(data_penguins)
```

💻 Use `nrow()` to find the number of the rows in the data `data_penguins`.

Sometimes we might just be interested in the values of a single column. To extract the values in a single column, we use the `$` operator, e.g. `data_penguins$id` will return all the values in the column called `id`.

💻 Run the commands `data_penguins$id` and `data_penguins$weight` in your R console to see what outputs you get (hint: you can use `head(data_penguins$id)` to only see the first few values).

Data types

Objects in R are assigned a data type, which tells R how to handle the object.

The simplest data types that you will commonly come across are:

- *character* i.e. text, e.g. `"f"` or `"London"`
- *numeric* and *integer*, e.g. `2L`, `1:10`, `2.1`, `pi`
- *date* e.g. `Sys.Date()`
- *logical* i.e. `TRUE` or `FALSE`

To check what data type an object is, use the function `class()`.

```
class(data_penguins$id)

class(data_penguins$weight)
```

❓ What data type is the column “length” in the `data_penguins` data? ❓ What data type is the column “sex” in the `data_penguins` data?

💻 Use the function `table` to compute the number of penguins classified as MALE and FEMALE:

```
table(data_penguins$sex)
```

💻 Combine the functions `prop.table` and `table` to compute the proportion of penguins classified as MALE and FEMALE:

```
prop.table(table(data_penguins$sex))
```

More complex data types can be used to combine multiple bits of data. For instance, objects that contain several elements are called data structures. There are three main types of data structure: - *vectors*: containing one variable per element (created by the function `c()`, like when we created `a` and `b` in the previous section). - *matrices*: containing several variables per element (all variables have the same type). - *data frames*: containing several variables per element (variables can have different types).

`data_penguins` contains several variables, of different types (some are characters, others are numeric), it is therefore a data frame. Columns of data frames are vectors:

❓ What would happen if `data_penguins` was a matrix? (hint: try running `head(as.matrix(data_penguins))`).

Other data types, such as *factor* and *list*, can be used and are not covered in this practical.

Basic operations with numeric data

With numeric data, we can calculate different summary statistics, such as the maximum and minimum values, the mean value,and the median value.

Mean and median

We can calculate the mean value with the function `mean()`.

💻 Try running the below command to calculate the mean weight:

```
# Calculate the mean weight
mean(data_penguins$weight)
```

This will return NA - not very helpful! This happens because there are missing values (NA values) in `weight` - you can see these by inspecting your data again. If there are any missing values in your data, `mean()` (and many other functions) will simply return NA.

To remove these missing (NA) values before calculating the mean, we include the argument `na.rm = TRUE`:

```
# Calculate the mean weight
mean(data_penguins$weight, na.rm = TRUE)
```

❓ What happens if you change this so that `na.rm = FALSE`?

By default, the value of the argument `na.rm` (standing for NA remove) in `mean()` is set to `FALSE`, so running `mean(data_penguins$weight, na.rm = FALSE)` is equivalent to `mean(data_penguins$weight)`.

Functions can have many arguments to tackle specific situations (e.g. how to deal with unreported entries). In order to read the description of the expected arguments and outputs of a function, you can use the operator `?`: e.g. `?mean`. It will open a helpfile, which will contain the list of all compatible arguments, and their default value (if any).

❓ What arguments are compatible with the function `mean` ?

We can calculate other statistics, too. We can calculate the median value with the function `median()`:

```
# Calculate the median weight
median(data_penguins$weight, na.rm = TRUE)
```

Note that we also have to include the `na.rm = TRUE` argument so that missing (NA) values are removed before the median is calculated.

💻 Calculate the mean and median wing length of the penguins in `data_penguins`. You can remind yourself of the other column names by looking at the data, or checking with the function we introduced earlier.

Minimum and maximum

We can calculate the minimum and maximum values with the functions `min()` and `max()`, respectively:

```
# Calculate the minimum weight
min(data_penguins$weight, na.rm = TRUE)

# Calculate the maximum weight
max(data_penguins$weight, na.rm = TRUE)
```

💻 Calculate the minimum and maximum wing length size for the penguins in `data_penguins`.

Standard deviation and inter-quartile range

We can calculate the standard deviation with `sd()` and the inter-quartile range (IQR) with `IQR()`:

```
# Calculate the standard deviation of weight
sd(data_penguins$weight, na.rm = TRUE)

# Calculate the IQR of weight
IQR(data_penguins$weight, na.rm = TRUE)
```

💻 Calculate the standard deviation and inter-quartile range for one of the other numeric columns in the `data_penguins` data.

A quick way to generate summary statistics for all columns in a data frame is to use the function `summary`:

```
# Show the summary of each column in data_penguins
summary(data_penguins)
```