# R Workshop: Basics

*Helio, Giedre and Amy*

*30/10/2019*

## 1. Starting you R session

First, create a folder on your desktop and call it something sensible (e.g. "R_Workshop". This will become your working directory later. The reason for having a working directory is that otherwise R will save working files into system directory, which makes them hard to find.

Open R studio. Explore the sections you can see on your screen. What does each of them do?

R is open source, which means that many people contribute to it by creating packages. The first package we will install is called R Markdown, which allows to generate a formatted file with figures and maintain version control. To install R Markdown enter the below to your Console:

```
# To install a package enter the below command
install.packages("rmarkdown")

# To load the package enter
library(rmarkdown)
```

Once this is done, you can go to File > New file and create a R Markdown document. Select the default output to be a Word document.

Save it in the folder you created by going to File > Save as. Pick a sensible name (e.g. My_R_Workshop).

Then you can set the folder you have created as your working directory easily by going to Session > Set Working Directory > To Source File Location

Explore the R Markdown file R created for you.

After this, click on the Knit (Make sure you save your Rmd first) at the top of this window.

Go to your folder and open the word document R created for you. Have a look at what corresponds to what in the Markdown and the Word document. Notice what corresponds to headings, normal text and code in particular. Feel free to delete everything below line 11.

Now you're all set to start!

### 1.1. Data types and data structures

R handles the following types of data:

1. Character: e.g. "a", "Hello!"
2. Numeric: e.g. 3, 1235
3. Integer: e.g. 3L
4. Logical: e.g. TRUE, FALSE
5. Complex: e.g. 2 + 4i

You can explore the types of data by typing something in brackets of the following commands and running them:

```r
class("Hello") # tells you the type of data
typeof("Hello") # similar to class() but more concerned how data is stored internally in R
length("Hello") # tells you how long it is
attributes("Hello") # checks if there is any meta data
```

Objects in R can be stored in the following data structures:

**1. Vector - the most basic of data structures in R, flat sequence of objects (e.g. "Hello" "it's" "me")**

```r
a <- 1:5 # makes a vector of digits from 1 to 5
a

# Check if this is really a vector
is.vector(a)

# Now try the below
b <- c("Hello", "Hi", "Good Morning", "Good afternoon", "Good evening")
b

# Check if this is really a vector
is.vector(b)
```

Note that if you try to create a vector which stores different types of data, it will force it to be a single type. E.g.

```r
c <- c(1, 2, "Hello")
c
typeof(c) # Notice how all your objects are now of  class character
```

**Now try creating a few more.**

**2. List - similar to vector, but can store nested data (i.e. a list within a list)**

```r
d <- list(a, a, list(a, b))
d

# Check if this is really a list
is.list(d) # yes!
```

**Now try creating a few more.**

**3. Matrix - a rectangular data structure that stored the same type of data (e.g. all integers)**

```r
e <- matrix(a, ncol = 3, nrow = 5, byrow = FALSE) # repeats a along the columns (byrow=FALSE) to create
e

# Alternatively, you can also use rbind() and cbind() commands to bind over rows and columns

# To check the dimensions of your matrix, use the following
ncol(e) # tells you how many columns there are
```

```r
nrow(e) # tells you how many rows there are
dim(e) # tells you how many rows and columns there are
# All of these also work on data frames
```

**Now try creating a few more.**

**4. Data frame - one of the most useful data structures, you can store your data using these as it can handle all types of data. E.g.:**

```r
f <- data.frame(cbind(e, b)) # binds a matrix e and a vector b along columns
f

# Check if f is a data frame
is.data.frame(f)
```

Note that the number of rows and columns should be equal, if not, you get an error (although R still tries it's best to give you a data frame by repeating vector b). Try creating a shorter vector (e.g. length = 3), binding it with a and notice what happens.

But if you create a longer vector and try again, it works fine

```r
g <- data.frame(cbind(a, b))
g

# You can also add a vector with missing values (NA). This is useful, because real life data normally h
h <- c(2, NA, 1, NA, 3)
g <- cbind(g, h)
g

# You can rename the columns:
names(g) <- c("Number", "Word", "Rating")
g

# And access particular columns using their names
g$Word

# Or coordinates (using format: dataFrame[rowNumber, columnNumber]). E.g.
g[1, c(2,3)]
```

**Now try creating a few more data frames and accessing columns and rows.**

**5. Factors - allows you to encode values and their labels**

```r
# Create another vector, which has factor labels to describe how formal the words in g$Word are, e.g.
i <- data.frame(c(1, 1, 2, 2, 2))

# Rename it
names(i) <- "Formality"

# Add it to dataframe g
g <- cbind(g, i)
```

```
g$Formality <- factor(g$Formality, levels = c (1, 2), labels = c("Informal", "Formal"))
g
```

**Now try creating a few more.**

You can save the data frame you have created to your working directory. E.g.

```
# This will be a tabs delimited text file
write.table(g, "My_data_frame.txt", sep="\t")
```

You can also import it back to R. E.g.

```
# This will import the data frame and set the first row as a header
My_data_frame <- read.delim("My_data_frame.txt", header=TRUE)
attach(My_data_frame)
```

## Using R as a calculator

You can perform basic calculations with R. Use +, -, /, *, sqrt(), ^2, log10() and see what happens. E.g.

```
2+1
```

You can also perform the same calculations with vectors. E.g.

```
a + 1
a * a
```

**Try some more calculations.**

However, notice how if one of the vectors is shorter, R recycles values, but only if the shorter vector is a multiple of the longer vector

```
# Create a longer vector
long <- 1:10
# Try adding them together
a + long
```

**Try some more calculations (including when shorter vector is of length 3).**

## Functions

You can use functions to help you with calculations within data structures. E.g.

```
apply(e, 1, mean) # apply calculates means of rows(1) of data frame f
```

There are a lot of packages in R that contain other very useful functions, such as for data wrangling, plotting, statistical tests. You can find most things that you need just by using Google. How to work these functions can be found in their documentation. If you have already installed the package you want and you do not know how to use a function, you can check what the function does by typing in ?function_name. E.g.

```
?cbind
```

You can also create functions of your own. This is incredibly useful if you are re-using the same bit of code in your script only changing a few things.

Basic structure of a function is

function_name <- function(variables){ do_this_with_variables }

E.g.

```r
# Create a function that adds two integers
my_function <- function(a, b){
c <- a + b
print(c)
}

# Try it out
my_function(2, 3)
```

**Install a package stats and make a function that calculates a mean and a standard deviation of a numeric vector of your choice and all columns of a matrix/data frame of your choice.**

## Logical statements

Logical operators in R look like this:

! stands for NOT & and && stand for AND (both statements have to be true to have TRUE) | and || stand for OR (one of the statements has to be true to have TRUE)

At the same time you will need the below too

!= stands for NOT EQUAL == stands for EQUAL "> more than < less than

Try making some statements that would give you TRUE or FALSE. E.g.

```r
a <- 5

3 + 1 != a

3 + 2 == a

3 + 2 == a | 3 + 1 == a

3 + 2 == a & 3 + 1 == a
```

## If statements

The basic structure of an if statement is shown below:

if (test_expression==TRUE) { do_this } else {do_this}

For example, try to make an if statement that prints "Well done!" if n is equal to or is more than 5 and "Better luck next time!" otherwise. E.g.

```r
n=1

if (n>=5){
  print("Well done!")} else {
  print("Better luck next time!")}
```

**Now try creating one yourself.**

## For loops

The basic structure of a for loop is shown below

for (number in from:to){ do_this }

For example, create a loop that loops from 1 to 10 and prints “Well done!” if n is equal to or more than 5 and “Better luck next time!” otherwise.

```
for (n in 1:10){
  if (n>=5){
  print("Well done!")} else {
  print("Better luck next time!")}
}
```

**Now modify it using “else if” so that it loops from 1 to 15 and prints “Better luck next time!” in d is between 1 and 5, “Good effort!” when 1 is between 6 and 10 and “Well done!” if d is equal to or more than 11.**