Beginner session in R

## Aim of the workshop

The aim of this session is to give you an overview of R, how the basics work, such as understanding different R objects, how to read in data, what different packages are and how to install them.  
- What is R and RStudio - What are functions - Different objects in R - What are packages - Tidy code - R help and resources

## What is R?

* R is free programming language for statistical analysis and data visualisation mainly
* It’s open source - anyone can add new functionality
* You type in commands to do things, rather than point and click like in excel
* Because everyone can add new ways of doing stuff, there is often more than one way to do the same thing in R

## What does R look like?

* There are 2 main programmes: R and Rstudio
* R is the base programme
* RStudio is more user-friendly, and therefore many people use it
* When you start Rstudio the screen is divided into 4 blocks/windows, as we will explain

R can be used as a calculator. Type in 3 + 5, or something like that, into the console.

## What are functions and packages?

R uses functions, which do certain things to the information you pass to them. Functions are always followed by brackets. For example, type in sum(2, 8, 3). This function does what it says! To find out about a function, type in ?>functionname<, and a help file will show up.

It is possible to write your own functions in R. Many people who write their own functions will save them in what is called a package. This means you can install the package, and use their functions. This really extends the capabilities of R, and we will look at installing packages later in this session.

## What are objects?

You can also save the numbers we used in our calculation, for example no <- sum(2, 8, 3). Now type in no.

In R, <- replaces =. While = works in some instances, it does weird things in others - best avoid it!

These entities that we can create in R are called objects. They are stored in the R environment. Have a look whether no appears in the window called Environment.

## Naming objects

R recognises each object by the name it is given, which can be made up of letters and numbers. For example: type in no + 3. R recognises no as the number 8.

* **R is case-sensitive so fish and Fish are two different objects!**
* Few symbols are allowed apart from ’\_’ and ‘.’
* Avoid object names that are functions in R e.g. sum

## Data types in R: Vectors

A vector is a collection of different elements which can hold different types of data.

*Numeric vector:*

a1 <- c(1, 2, 5.3, 6, -2, 4)  
a1

## [1] 1.0 2.0 5.3 6.0 -2.0 4.0

str(a1)

## num [1:6] 1 2 5.3 6 -2 4

Note that c() is a function. It tells R to expect a whole set of numbers, letters, or whatever you input.

str() is also a function. It gives the structure of our object, and is very useful to get an idea of what object we are working with.

There are numeric vectors, as well as integers.

*Character vector:*

b1 <- c("one","two","three")   
b1

## [1] "one" "two" "three"

str(b1)

## chr [1:3] "one" "two" "three"

This is very similar, except this time you are inputting characters. It won’t work without the inverted commas!

*Logical vector:*

c1 <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE)   
c1

## [1] TRUE TRUE TRUE FALSE TRUE FALSE

str(c1)

## logi [1:6] TRUE TRUE TRUE FALSE TRUE FALSE

## Data types in R: Matrix

A matrix is a vector with added dimensions - it must have columns and rows. All columns must have the same length and type of data, such as numeric, or characters.

y <- matrix(1:20, nrow = 5, ncol = 4)   
# nrow and ncol determine the numbers of rows and columns   
y

## [,1] [,2] [,3] [,4]  
## [1,] 1 6 11 16  
## [2,] 2 7 12 17  
## [3,] 3 8 13 18  
## [4,] 4 9 14 19  
## [5,] 5 10 15 20

str(y)

## int [1:5, 1:4] 1 2 3 4 5 6 7 8 9 10 ...

s\_means = matrix(c(-14, -15.1, -11.03, 3.06, 7.05, 13.72), ncol = 2, nrow = 3)  
s\_means

## [,1] [,2]  
## [1,] -14.00 3.06  
## [2,] -15.10 7.05  
## [3,] -11.03 13.72

str(s\_means)

## num [1:3, 1:2] -14 -15.1 -11.03 3.06 7.05 ...

## Data types in R: Dataframes

They are more general than a matrix and can have columns with different data types. If you have your own data, this is probably what you will get once you read the file into R.

dat <- data.frame(id = letters[1:3], x = 1:3, y = 4:6)  
dat

## id x y  
## 1 a 1 4  
## 2 b 2 5  
## 3 c 3 6

str(dat)

## 'data.frame': 3 obs. of 3 variables:  
## $ id: Factor w/ 3 levels "a","b","c": 1 2 3  
## $ x : int 1 2 3  
## $ y : int 4 5 6

# Have a good look at the code above, and the output you get.

## Data types in R: List

Lists are ordered collections of objects and they can have different modes in one list

w <- list(name="Fred", mynumbers = a1, mymatrix = y, age = 5.3)  
str(w)

## List of 4  
## $ name : chr "Fred"  
## $ mynumbers: num [1:6] 1 2 5.3 6 -2 4  
## $ mymatrix : int [1:5, 1:4] 1 2 3 4 5 6 7 8 9 10 ...  
## $ age : num 5.3

## Set working directory

If you want to import an excel file with your data in, rather than typing it in by hand you need to tell R which folder contains your file. You do this by setting the working directory. But before you do this you need to create a new folder and download and save the example data file in this new folder.

First of all create a new folder called >come\_up\_with\_a\_witty\_name< on your laptop.

There are two different ways to set the working directory:

1. Use the main menu to navigate to the right folder

On the RStudio main menu click on:

Session > Set Working Directory > Choose Directory

Navigate to the Rsession folder you have just created and click once on the folder so its highlighted in blue, then click on the Select folder button.

You can check you have the right folder set as your working directory:

getwd()

You should see:>my\_file\_path<

1. You can tell R to set the working directory using the function setwd()

On the top left hand window (in your R script) type:

setwd(>my\_file\_path<)

Be aware that you need to type the exact file path otherwise R won’t be able to understand it.

If you want to know the file path you can also go to windows explorer, find your folder and click once on the file path bar on the top of the window and it will show the file path. Just be careful: if you copy this, you need to change the backslashes \ to forward slashes /.

### Read in data

Note: R likes using .csv files rather than normal excel spreadsheets (.xlsx files). So please save the files you want to import into R as .csv files.

Also make sure they are tidy - e.g. data in two columns with no gaps in between, no random plots, random bits of writing etc., otherwise R won’t know how to interpret it.

We will practice loading data into R now. Copy the following file into Excel:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
| 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 5.0 | 3.6 | 1.4 | 0.2 | setosa |
| 5.4 | 3.9 | 1.7 | 0.4 | setosa |
| 4.6 | 3.4 | 1.4 | 0.3 | setosa |
| 5.0 | 3.4 | 1.5 | 0.2 | setosa |
| 4.4 | 2.9 | 1.4 | 0.2 | setosa |
| 4.9 | 3.1 | 1.5 | 0.1 | setosa |

Now save the file in the R session folder you created. Call it >something< and save it as type CSV (Comma delimited).

To read in your file you use the function read.csv in R:

my\_data <- read.csv(">something<.csv")   
# here the file will be called my\_data in R, and as the working directory will be the folder in which your data file is saved, R will find it there

You can also save files back into your working directory with the function write.csv, if you have changed them in R:

write.csv(my\_data, ">somethingelse<.csv")   
# the first bit in the bracket is the name of the dataset in R, the second bit is the name of the new csv file you want to save

## Some basic outputs from a dataset

Apart from the str() function, there are some other functions we can use to get an initial idea of our data. We can use summary() for the whole dataset. For continuous variables, this will show the minimum and maximum value, the quartiles and the mean. For categorical variables, it will show the different categories and a count of each one. We will use a built-in dataset called iris:

summary(iris)

## Sepal.Length Sepal.Width Petal.Length Petal.Width   
## Min. :4.300 Min. :2.000 Min. :1.000 Min. :0.100   
## 1st Qu.:5.100 1st Qu.:2.800 1st Qu.:1.600 1st Qu.:0.300   
## Median :5.800 Median :3.000 Median :4.350 Median :1.300   
## Mean :5.843 Mean :3.057 Mean :3.758 Mean :1.199   
## 3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd Qu.:1.800   
## Max. :7.900 Max. :4.400 Max. :6.900 Max. :2.500   
## Species   
## setosa :50   
## versicolor:50   
## virginica :50   
##   
##   
##

We can also get the summary for just one variable. We need to tell R in which dataset to look for it, and then add the column name after a dollar sign:

summary(iris$Sepal.Length)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 4.300 5.100 5.800 5.843 6.400 7.900

This also works for other functions, such as mean:

mean(iris$Sepal.Length)

## [1] 5.843333

To get a better idea of your dataset, you can also use the head function, which will show the first 6 rows of the dataset, unless otherwise specified in the code:

head(iris)

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 1 5.1 3.5 1.4 0.2 setosa  
## 2 4.9 3.0 1.4 0.2 setosa  
## 3 4.7 3.2 1.3 0.2 setosa  
## 4 4.6 3.1 1.5 0.2 setosa  
## 5 5.0 3.6 1.4 0.2 setosa  
## 6 5.4 3.9 1.7 0.4 setosa

head(iris, 10)

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 1 5.1 3.5 1.4 0.2 setosa  
## 2 4.9 3.0 1.4 0.2 setosa  
## 3 4.7 3.2 1.3 0.2 setosa  
## 4 4.6 3.1 1.5 0.2 setosa  
## 5 5.0 3.6 1.4 0.2 setosa  
## 6 5.4 3.9 1.7 0.4 setosa  
## 7 4.6 3.4 1.4 0.3 setosa  
## 8 5.0 3.4 1.5 0.2 setosa  
## 9 4.4 2.9 1.4 0.2 setosa  
## 10 4.9 3.1 1.5 0.1 setosa

# this will show the first 10 rows

The same works for the last 6 rows with the tail function:

tail(iris)

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 145 6.7 3.3 5.7 2.5 virginica  
## 146 6.7 3.0 5.2 2.3 virginica  
## 147 6.3 2.5 5.0 1.9 virginica  
## 148 6.5 3.0 5.2 2.0 virginica  
## 149 6.2 3.4 5.4 2.3 virginica  
## 150 5.9 3.0 5.1 1.8 virginica

tail(iris, 10)

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 141 6.7 3.1 5.6 2.4 virginica  
## 142 6.9 3.1 5.1 2.3 virginica  
## 143 5.8 2.7 5.1 1.9 virginica  
## 144 6.8 3.2 5.9 2.3 virginica  
## 145 6.7 3.3 5.7 2.5 virginica  
## 146 6.7 3.0 5.2 2.3 virginica  
## 147 6.3 2.5 5.0 1.9 virginica  
## 148 6.5 3.0 5.2 2.0 virginica  
## 149 6.2 3.4 5.4 2.3 virginica  
## 150 5.9 3.0 5.1 1.8 virginica

## Changing the type of variable

If variables are not in the right format, it can mess up our analysis or our plots, as R deals differently with different types of variables. Sometimes R does not read the variables in the right way, or we need to change the variable type for some reason. Once we have checked whether our variables are in the correct format, we can change those that are not:

iris$Species<-as.character(iris$Species)  
  
# we can check what type of variable we have in different ways:  
str(iris$Species)

## chr [1:150] "setosa" "setosa" "setosa" "setosa" "setosa" "setosa" ...

is.character(iris$Species)

## [1] TRUE

In this case, species should really be a factor, so we will change it back to that:

iris$Species<-as.factor(iris$Species)

In the same way, we can change variables to integers (as.integer) or continuous variables (as.numeric).

## How to install and load packages

R has a huge library of additional packages. At some point you will most likely need to use some of these packages - they might be able to run some analyses which will save you from writing lots of code, contain some datasets that are of interest to you, or help with organising or displaying your data, for example. Some packages are very specific and will only be of interest to a few, whereas others are widely used. For a full list see <https://cran.r-project.org/web/packages/available_packages_by_name.html>

As with all programmes on your computer, you need to install a package the first time that you use it, and load it every following time. To install a package, you can either use the code

install.packages("dplyr") # in this case, we are installing a package called dplyr

Or choose the package through the menu: Tools > Install packages..., then you can type the name of the package.

Once you have installed your package, you can load it:

library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

*Every time you start R, you will have to load the package again if you want to use a function in that package!*

To find out about all the different functions in a package, you can usually find a detailed description of all functions in a package (called vignette) online, just google R package >packagename<. For dplyr, this will get you the following pdf: <https://cran.r-project.org/web/packages/dplyr/dplyr.pdf>

dplyr is a useful package for manipulating data. We will use a dataset called airquality for some basic functions with dplyr. Some of the easiest to use functions in the package are filter and select.

filter will retain or remove rows in your dataset, according to conditions you set:

temphigh <- filter(airquality, Temp > 79)   
# this creates a new dataset called temphigh, which are all values from dataset airquality where temperature is above 79  
  
templow <- filter(airquality, Temp <= 79)   
# this creates a new dataset called temphigh, which are all values from dataset airquality where temperature is below or at 79

Have a look how many observations there are in airquality, temphigh and templow.

We can also examine rows that contain NA values, or exclude rows that contain NA values. The structure is slightly different when it comes to including or excluding NAs:

summary(airquality)

## Ozone Solar.R Wind Temp   
## Min. : 1.00 Min. : 7.0 Min. : 1.700 Min. :56.00   
## 1st Qu.: 18.00 1st Qu.:115.8 1st Qu.: 7.400 1st Qu.:72.00   
## Median : 31.50 Median :205.0 Median : 9.700 Median :79.00   
## Mean : 42.13 Mean :185.9 Mean : 9.958 Mean :77.88   
## 3rd Qu.: 63.25 3rd Qu.:258.8 3rd Qu.:11.500 3rd Qu.:85.00   
## Max. :168.00 Max. :334.0 Max. :20.700 Max. :97.00   
## NA's :37 NA's :7   
## Month Day   
## Min. :5.000 Min. : 1.0   
## 1st Qu.:6.000 1st Qu.: 8.0   
## Median :7.000 Median :16.0   
## Mean :6.993 Mean :15.8   
## 3rd Qu.:8.000 3rd Qu.:23.0   
## Max. :9.000 Max. :31.0   
##

# Ozone contains 37 rows with NA values  
   
airqualityNA <- filter(airquality, is.na(Ozone))   
# In this case we keep those rows where Ozone is NA  
  
summary(airqualityNA)

## Ozone Solar.R Wind Temp   
## Min. : NA Min. : 31.0 Min. : 1.70 Min. :56.00   
## 1st Qu.: NA 1st Qu.:131.0 1st Qu.: 8.00 1st Qu.:75.00   
## Median : NA Median :194.0 Median : 9.70 Median :79.00   
## Mean :NaN Mean :189.5 Mean :10.26 Mean :77.92   
## 3rd Qu.: NA 3rd Qu.:261.5 3rd Qu.:12.60 3rd Qu.:84.00   
## Max. : NA Max. :332.0 Max. :16.60 Max. :93.00   
## NA's :37 NA's :2   
## Month Day   
## Min. :5.000 Min. : 1.00   
## 1st Qu.:6.000 1st Qu.:10.00   
## Median :6.000 Median :15.00   
## Mean :6.351 Mean :16.65   
## 3rd Qu.:7.000 3rd Qu.:25.00   
## Max. :9.000 Max. :30.00   
##

airqualitynoNA <- filter(airquality, !is.na(Ozone))   
# Often we might want to remove NAs from our dataset, not keep them - we do this by adding ! in front of is.na. Here we keep rows if Ozone is not NA  
  
summary(airqualitynoNA)

## Ozone Solar.R Wind Temp   
## Min. : 1.00 Min. : 7.0 Min. : 2.300 Min. :57.00   
## 1st Qu.: 18.00 1st Qu.:113.5 1st Qu.: 7.400 1st Qu.:71.00   
## Median : 31.50 Median :207.0 Median : 9.700 Median :79.00   
## Mean : 42.13 Mean :184.8 Mean : 9.862 Mean :77.87   
## 3rd Qu.: 63.25 3rd Qu.:255.5 3rd Qu.:11.500 3rd Qu.:85.00   
## Max. :168.00 Max. :334.0 Max. :20.700 Max. :97.00   
## NA's :5   
## Month Day   
## Min. :5.000 Min. : 1.00   
## 1st Qu.:6.000 1st Qu.: 8.00   
## Median :7.000 Median :16.00   
## Mean :7.198 Mean :15.53   
## 3rd Qu.:8.250 3rd Qu.:22.00   
## Max. :9.000 Max. :31.00   
##

Have a look at the different datasets, and make sure R is doing what you think it is doing.

The select function can be used to include or exclude columns from the dataset:

airquality2 <- select(airquality, -Ozone)   
# this will make a new dataset called airquality2, based on airquality, without the variable Ozone  
  
str(airquality2)

## 'data.frame': 153 obs. of 5 variables:  
## $ Solar.R: int 190 118 149 313 NA NA 299 99 19 194 ...  
## $ Wind : num 7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...  
## $ Temp : int 67 72 74 62 56 66 65 59 61 69 ...  
## $ Month : int 5 5 5 5 5 5 5 5 5 5 ...  
## $ Day : int 1 2 3 4 5 6 7 8 9 10 ...

airquality3 <- select(airquality, Ozone, Solar.R)   
# this will make a new dataset called airquality3, based on airquality, with only the variables Ozone and Solar.R  
  
str(airquality3)

## 'data.frame': 153 obs. of 2 variables:  
## $ Ozone : int 41 36 12 18 NA 28 23 19 8 NA ...  
## $ Solar.R: int 190 118 149 313 NA NA 299 99 19 194 ...

Again, examine the different datasets, and check that R is doing what you think it is doing.

## Saving R files

It is important to save your code - if you find a mistake in your dataset, or you need to run some analysis again, it will be much quicker to do if you have your code saved.

To save your R script, go to File > Save as... and then find the folder that you want to save your script in.

## Tidy code

As a beginner, you have a unique opportunity to code in a tidy way from the start! For some reason, a lot of R courses do not cover this, but it is really important for readibility (for you AND others). Here are examples of what you should aim to do:

* Always have a space after a comma
* Always have spaces before and after <-
* Avoid capital letters (because R is case sensitive, this makes life so much easier)
* Give your objects meaningful names. Do not call everything data for example

BAD code:

Data<-c(19,15,17,12,15,18,22) #these are summer temperatures in Newcastle

GOOD code:

# Change the BAD code above to some nice, tidy code!

It is also *really important* that you annotate what you do, so if you come back to your code at a later date you know what you were doing. As a beginner, this is particularly important - if you have a really good memory, you might remember what all the functions do, but if you are like me you won’t. Therefore, annotate as much as possible :)

# if you add a hash, R won't execute what you write after it until the next line. This way you can annotate your code

It is also possible to add subheadings into your code. This is useful for long scripts in particular.

# >Your amazing heading name<, followed by ####

* In your script window, at the bottom, your heading should appear next to the orange # sign
* If you click on a heading, it takes you to that part of your script

There are some great resources for ensuring your code is beautiful, for example by the British Ecological Society <https://www.britishecologicalsociety.org/wp-content/uploads/2017/12/guide-to-reproducible-code.pdf>

## Solving problems in R/ help

Sometimes it is difficult to know how exactly you need to write your code for a certain function, and you might get error messages. These can be difficult to interpret, especially if you are new to R. You can get more information on how to set up your code within a function by finding an R help file:

?summary  
# this will tell you more about the summary function

The help files are very useful, and the more you use R, the better you will understand them. Often it is easiest to look at the example code at the end.

You can google error messages, and are likely to find that other people have encountered them too. It is a bit of a learning curve to understand the responses, so the more you practice, the easier it gets.

## Useful resources

### Online

There is lots of information online about all kinds of R related problems, tutorials and so on. Here are a few that are particularly useful if you are new to R:

* R for data science: a book about how to arrange your data, data visualisation, programming and model building in R. It is available for free online: <https://r4ds.had.co.nz/>
* R bloggers: each post covers a topic with some example code. This can be handy if you have a specific R-related problem, want to know some basic information about a new package you are using, etc.: <https://www.r-bloggers.com/>
* Swirl stats: Learn R, in R - this is an interactive tool with different modules that teach R within an R package: www.swirlstats.com
* stackoverflow: People can post R-related questions, and more experienced users help to find solutions. The examples provided in the questions need to be reproducible however. Often the questions you may have will already have been asked and answered on stackoverflow, so it is worth checking whether you can find a solution already: <https://stackoverflow.com/>

### Checklist

Other than those helpful resources here is a checklist of what to do if you are stuck with a problem in R:

* First of all don’t panic or get frustrated! Everyone will have a problem at some point in time if they are learning something new
* Have a sanity check and look for mistakes in your code e.g. spelling mistakes, an object isn’t in your data environment, you haven’t called library() before using a package.
* Google it! There is somewhat an art to wording your question in google to get the type of answers you are looking for but this will get better with practise. Chances are someone else will have definitely faced the same problem as you, even if it’s not exactly in the same format and the answer is somewhere on the internet.
* Can you find an example of what you want where it works? Even if it’s a simpler example understanding the basics of it might help you work out what’s wrong. Have a close look at that example, what exactly the data they used was, what format it was etc.
* Sometimes it’s just better to walk away (for now). If you have been struggling for a couple of hours it’s sometimes really good to just walk away from it and start again another day when you have a fresh pair of eyes (and hopefully some renewed patience). Or even just going for a small walk can help you get some clarity on the problem.
* If you really can’t suss it ask a question on stack overflow (which can be quite intimidating as you need to follow certain rules e.g. have a small chunk of data which is reproducible)
* People trying to help you, particularly when it’s someone that can’t sit with you and actually see your computer (i.e. someone on the internet) often find it useful to have a reproducible example, creating one of these in itself can be quite challenging but have a look at the ‘reprex’ package, it helps you create a reproducible example snippet of your code which might help you to better understand your error/ problem and will help those people trying to help you