MATH10282 Introduction to Statistics
Semester 2, 2020/2021
Examples Class 1: Introduction to R

Introduction

These sessions aim to provide an introduction to using the statistical computing package R. We will learn how to read data into R, perform various calculations and obtain summary statistics for data. **You should work through the notes given and seek clarification and help when you need it from the staff or postgraduate students in the room.**

R is an open-source statistical environment, originally released in 1997. It is an implementation of the S language, which was developed in the 1980s. There is also a commercial, if somewhat expensive, version of S called S-PLUS sold by Insightful Corporation. R provides similar functionality to S-PLUS but it is free.

Several textbooks discuss the use of R for statistical data analysis. A good introduction is given by:

Crawley, M. J. (2005) Statistics, An Introduction using R. (Wiley)

which is available in the university library.

The command language for R is a computer programming language. The syntax is fairly straightforward and there are many built-in statistical functions. The language can be easily extended with user-written functions. R also has very good graphical facilities.

Obtaining and installing the R software

R is already available on the workstations in the Alan Turing Building. You can also download and install a copy of the latest version of R for free on your own computer from the Comprehensive R Archive Network (CRAN) web site,

`http://www.stats.bris.ac.uk/R/`

This site contains lots of information about R including FAQ's, manuals, and links to relevant mailing lists.

Once on the CRAN site, select 'Download R for Windows' (if that is the operating system you use), choose 'base' and then 'Download and install R 3.4.3'. There are also versions for Linux and Mac OSX operating systems. User-contributed packages of functions can be installed later as required.

Under 'Documentation' you can download 'An Introduction to R' by W. N. Venables, D. M. Smith and the R Development Core Team (2006). This gives a clear introduction to the language, and information on how to use R for statistical analyses and graphics. This manual is also available in the R software via the menu:

Help → Manuals (in PDF)→ An Introduction to R.

Getting started

In Room G.105 of the Alan Turing Building, you can start up the R software by opening the Start menu and choosing

All Programs → R → Rx64 3.3.0

When the program has loaded you should see the R desktop window (see the figure below). The subwindow in the top left is called the command window, or console, and R is used by typing appropriate commands here. At the beginning of the command line you will see the R prompt `>`. To interact with R, type your command following the prompt and press return. Depending on the command, R will either display the results of your command on the next line, open a new window, or simply create a new prompt if your command does not result in any output. If a command is too long to fit on a line, R automatically includes a `+` to denote continuation.

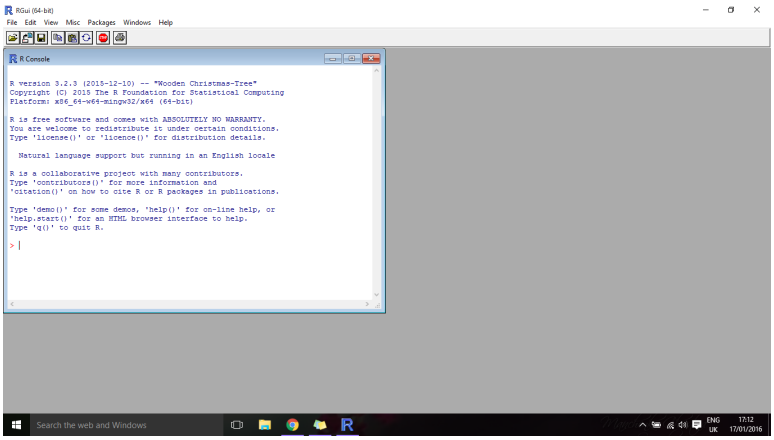


Figure: The R desktop window on start up.

Useful tip: previous commands can be accessed using the up and down arrow keys on the keyboard. This can save time typing as previous commands can be edited and reused.

R is a very powerful computing package and I hope this session gives you a flavour of how it works and what it can do. This is the same statistical software that will be used in second and third year Statistics modules. It is also widely used in many areas in academia and industry.

Getting help

R has an excellent built-in help facility, accessible via the 'Help' drop-down menu at the top of the R window. Alternatively, to get information about a particular R function use

```
> help(function name)
```

replacing `function name` above with the name of the function in which you are interested, such as `length`, `seq`, `rep`, `var`, `sd`, `mean` etc. For example, for the first function type `help(length)`.

Using R

You should practice using R in this session by copying and typing the commands which are given following the `>` prompt in the notes below and then observing the output. There are also some exercises for you to try.

Simple calculations

We can use R as a calculator by typing an arithmetic command after the `>` prompt followed by the return key. Multiplication is denoted using `*`, division using `/`, and powers are denoted using `^`, e.g.:

```
> 33 + 24 - 8
[1] 49
> 25*20
[1] 500
> 2786/256
[1] 10.88281
> 4^3
[1] 64
> 2 + 4*20/10 # R operator precedence rules are conventional
[1] 10
> pi*10^2 # pi denotes the usual constant 3.14159..
[1] 314.1593
```

In each case, R returns the result as a vector of length 1. The `[1]` at the start of the line indicates that the next number is the 1st element of the vector.

Comments

The hash symbol `#` used above indicates that the rest of the line is a comment, and not part of the command. R does not attempt to execute comments. However, it is extremely useful to include comments to indicate what the different commands achieve.

Objects

Now we consider storing data in objects which can then be manipulated in R.

```
> a <- 23 # assigns the value 23 to the object "a".
```

Note that the operator `<-` indicates assignment. We can also use the `=` operator instead. Typing `a` results in R printing its value, i.e.

```
> a
[1] 23
```

We can perform arithmetic using the variable names, e.g.

```
> b <- 5
> b
[1] 5
> a*b
[1] 115
> a/b
[1] 4.6
> c<-10*a+b
> c
[1] 235
> ls()
[1] "a" "b" "c"
> rm(a, b, c)
> ls()
character(0)
```

Listing and removing objects

To display the names of all objects currently stored within R, use the command `ls()`. To delete objects from R, use the command `rm()` where the name(s) of the objects to be removed are placed inside the brackets, separated by commas. To remove everything you use the command `rm(list=ls())` but this is rather drastic.

Case-sensitivity

R commands are case sensitive, i.e. `x` and `X` are considered different. It is recommended that you primarily use lower case letters.

```
> x <- 5
> X
Error: object 'X' not found
```

Vectors

The objects `a`, `b` and `c` above are examples of vectors which happen to all be of length 1. To create a numeric vector with more than one element, use the `c` command:

```
> x <- c(1.4, 2.7, 4.9, 5.5, 7.8, 10.2, 14.2, 15.0, 16)
> x
[1]  1.4  2.7  4.9  5.5  7.8 10.2 14.2 15.0 16.0
```

The function `c()` combines its arguments (i.e. the numbers inside the brackets that are separated by commas) into a single vector. We can perform numerical operations on this new vector as follows:

```
> 2*x
[1]  2.8  5.4  9.8 11.0 15.6 20.4 28.4 30.0 32.0
> 2*x+1
[1]  3.8  6.4 10.8 12.0 16.6 21.4 29.4 31.0 33.0
> 1/x
[1] 0.71428571 0.37037037 0.20408163 0.18181818
[5] 0.12820513 0.09803922 0.07042254 0.06666667
[9] 0.06250000
```

Note that the commands act on all the elements of the vector. Some further examples of functions which can be applied to a vector `x` are:

```
> length(x) # computes the number of elements in x
[1] 9
> sort(x)   # sort the elements of x into ascending order
[1]  1.4  2.7  4.9  5.5  7.8 10.2 14.2 15.0 16.0
> sqrt(x)
[1] 1.183216 1.643168 2.213594 2.345208 2.792848
[6] 3.193744 3.768289 3.872983 4.000000
> sum(x)
[1] 77.7
> mean(x)
[1] 8.633333
> var(x)
[1] 30.0275
> sd(x)
[1] 5.479735
> sqrt(var(x))
[1] 5.479735
> s <- sqrt( sum( (x-mean(x))^2/(length(x)-1) ) )
> s
[1] 5.479735
> v <- s^2
> v
[1] 30.0275
> ls()
[1] "s" "v" "x"
```

Accessing the elements of a vector

We will now see how to access individual elements or subsets of elements in a vector by looking at some examples.

```
> x[3] # the 3rd element of x
[1] 4.9

> x[-3] # all but the 3rd element
[1]  1.4  2.7  5.5  7.8 10.2 14.2 15.0 16.0

> x[2:5] # the 2nd to 5th elements
[1] 2.7 4.9 5.5 7.8
```

```
> x[(length(x)-2):length(x)] # the last 3 elements
[1] 14.2 15.0 16.0

> x[c(2, 4, 8)] # the 2nd, 4th and 8th elements
[1]  2.7  5.5 15.0

> x[x>12] # all elements > 12
[1] 14.2 15.0 16.0

> x[x<2 | x>15] # elements <2 or >15
[1]  1.4 16.0

> x[x>6 & x<12] # elements >6 and <12
[1]  7.8 10.2
```

Logical vectors

As well as numerical vectors, we can also define **logical vectors**. These vectors have elements equal to either `TRUE` or `FALSE` and are generated by conditions, e.g.

```
> x
[1]  1.4  2.7  4.9  5.5  7.8 10.2 14.2 15.0 16.0
> x>5
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
> lv <- (x>5)
> lv
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Logical vectors provide another convenient way of 'subsetting' a vector, e.g. the following command returns the elements of `x` greater than 5:

```
> x[x>5]
[1]  5.5  7.8 10.2 14.2 15.0 16.0
```

The following logical operators can be used:

Operator	Meaning
<	less than
>	more than
<=	less than or equal to
>=	greater than or equal to
==	equal to
!=	not equal to
&	and
|	or
!	negation

Note that to check if `a` and `b` are equal, you should use `==` and not `=`. If `c1` and `c2` are logical vectors, then `c1 & c2` checks elementwise to see if `c1` and `c2` are both `TRUE`, e.g.

```
> c1 <- c(TRUE, TRUE)
> c2 <- c(TRUE, FALSE)
> c1 & c2
[1]  TRUE FALSE
```

Similarly, `c1 | c2` checks elementwise if either `c1` or `c2` (or both) is `TRUE`, e.g.

```
> c1 | c2
[1]  TRUE TRUE
```

Finally, `!c1` is the negation of `c1`, i.e. `TRUE` is changed elementwise to `FALSE` and vice versa, e.g.

```
!c1
[1] FALSE FALSE
```

We can also use logical vectors in ordinary arithmetic, in which case they are first automatically transformed into numeric vectors with `FALSE` becoming 0 and `TRUE` becoming 1. This can be useful for counting the number of times an event occurs, e.g.

```
> sum(lv)  # number of elements of x that are greater than 5
[1] 6
> sum(x>14) # number of elements of x that are greater than 14
[1] 3
```

Exercises

1. Let the vector $\mathbf{y} = (1, 2, 4, 12, 16)^T$ and $\mathbf{z} = (3, 4, 8, 9, 15, 18)^T$. Read the data into two vectors `y` and `z` in R. Before carrying out each of the following commands, try to guess the result first.

   (i) `y-3`

   (ii) `z*3 + 1`

   (iii) `median(y)` and `median(z)`.

   (iv) `sum(z[z>10])`

   (v) `mean(y[2:4])`

   (vi) `mean(z[-(4:6)])`.

   (vii) `y+z` (Why does this not work?)

   (viii) `sum(y>=4)` and `sum(y[y>=4])`

   (ix) `sum(z>5 & z<12)`

2. The following data are the closing prices (in pence) of a share in a particular company over a twelve day period:

$$38, 42, 44, 43, 46, 48, 52, 54, 50, 47, 48, 48$$

   (i) Enter the data into a vector `s` in R.

   (ii) Use R to find the minimum, maximum, range, mean and standard deviation of the twelve values.

   (iii) We find out that the first value was in fact not 38, and needs to be corrected to 48. Edit the data so that it is now correct and then recalculate the mean. Remember, we access the elements in a vector with `[]`.

   (iv) Based on the corrected data from part (iii), use R to find the percentage of days when the closing share price was less than 45 pence.

   (v) Use the function `diff` on the data in `s` and store the results in the vector `d`. What are the values stored in `d`? What is the length of `d`?

Reading data from a file

We will now read two data sets into R, as follows:

1. Download the data sets

   `https://minerva.it.manchester.ac.uk/~saralees/moses.txt`

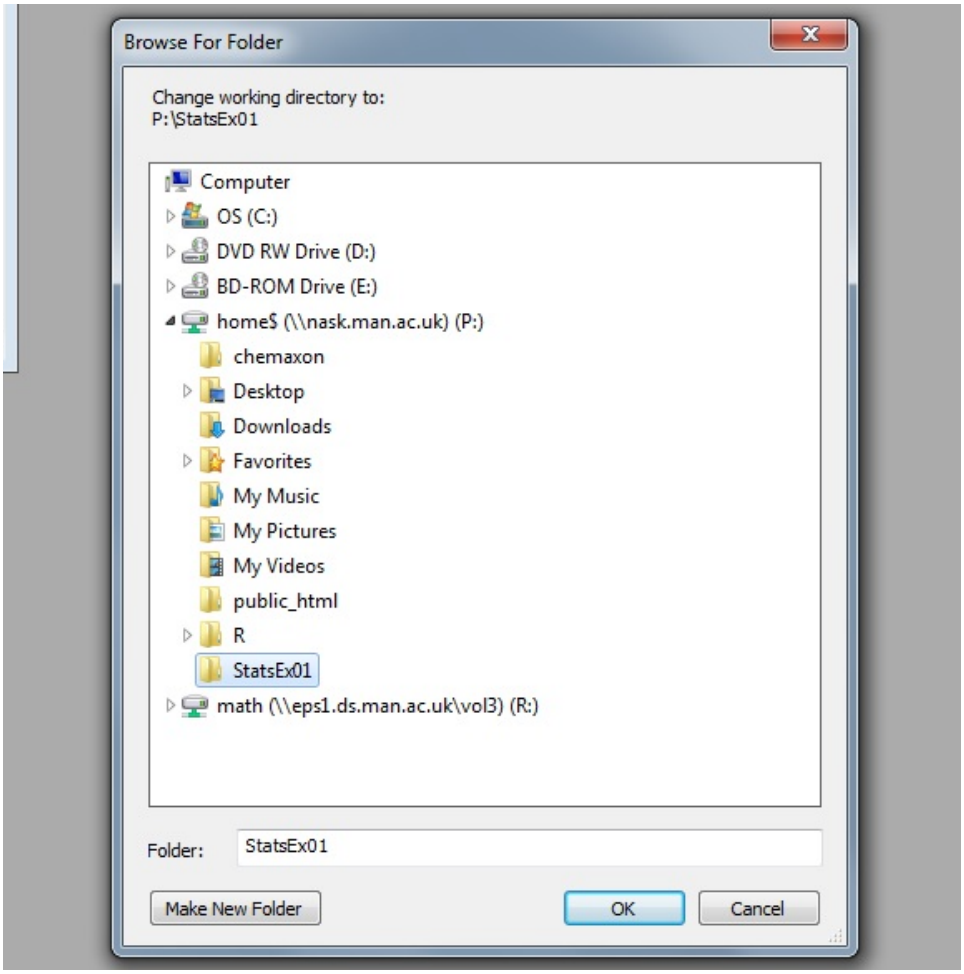
   `https://minerva.it.manchester.ac.uk/~saralees/opinion_poll_2015.txt`

To download these files, **right click** the corresponding link, select 'Save As...', and then navigate to an appropriate location in which to save the file. It is easiest to save them to either (a) `C:\Work`, (b) a folder in 'My Documents' or (c) a folder on a memory stick. For example, you could call the folder `StatsEx01` to indicate what it contains.

2. Go into R and **change the working directory** to the same folder in which you have stored the data. Do this by clicking on the menu item:

   'File → Change dir'

   and browsing to the relevant folder. For example, if you saved the data in 'StatsEx01' within 'My Documents' (this is also known as your P: drive), select this folder as follows:



3. We can now read the data using the `read.table()` command, e.g.

```
> moses <- read.table(file="https://minerva.it.manchester.ac.uk/~saralees/moses.txt",
                      header=TRUE)
```

   The argument `header=TRUE` indicates that the first row of the file contains the variable names. If it does not, then use the argument `header=FALSE`. Use `help(read.table)` to find out more information about this command.

The above command reads the data from the file `moses.txt` and stores it in an R object called `moses`. This data set contains the winning times of the American athlete Ed Moses in 122 competitive races of the 400m hurdles between 1976 and 1978. The times are in chronological order. To print out the contents of the file, enter the following:

```
> moses
```

The row numbers 1 to 122 in the first column are not in the file, but are added by R. The times can be extracted from the R object using `moses$time`. Alternatively, if you issue the command `attach(moses)` first, then this will make `moses` the active dataset within R and you can then refer to the time data using `time`.

- **Exercise:** For the Moses data, calculate the minimum, median and maximum times. What proportion of the times are less than 47.5s? What proportion are greater than 49.5s?

Categorical data

With categorical data, the variable of interest records which of a finite number of categories an individual belongs to. For example, a survey asked a small sample of 15 people whether or not they were part of an employer's pension scheme. The responses recorded were:

yes, yes, no, yes, no, no, yes, no, no, yes, no, yes, yes, no, no

We can again enter this data into R using the `c()` command and summarize them with the table command, i.e.

```
> response <- c("yes", "yes", "no", "yes", "no", "no",
   "yes", "no", "no", "yes", "no", "yes", "yes",
   "no", "no")
> response
 [1] "yes" "yes" "no"  "yes" "no"  "no"  "yes" "no"
 [9] "no"  "yes" "no"  "yes" "yes" "no"  "no"
> table(response)
response
 no yes
  8   7
```

Factors

It is common practice with categorical data to assign numerical values to each of the categories. For example, with the pension scheme data above, we could assign the number 1 to the category 'yes' and 0 to 'no'.

In R, categorical data should be declared as a **factor** whether it is recorded as characters or numerically.

```
> responsef <- factor(response)
> responsef
 [1] yes yes no yes no  no yes no
 [9] no  yes no yes yes no no
Levels: no yes
```

The `levels()` command allows us to identify the different levels/categories/values the factor can take.

```
> levels(responsef)
[1] "no"  "yes"
```

It is straightforward to recode the levels 'no' and 'yes' to be 0 and 1, as follows:

```
> levels(responsef) <- c(0, 1)
> levels(responsef)
[1] "0" "1"
> responsef
 [1] 1 1 0 1 0 0 1 0 0 1 0 1 1 0 0
Levels: 0 1
```

- **Exercise:** Use the `read.table` command to read the opinion poll data into R from the file `https://minerva.it.manchester.ac.uk/~saralees/opinion_poll_2015.txt`. Check the file first to see whether the first row contains the variable names. In the file, the preferences are coded as 1 (Conservative), 2 (Labour), 3 (Liberal Democrats), 4 (UKIP) and 5 (Other). Firstly, specify the data as factors and then recode the numeric values $1-5$ into character values. Use the `table` command to summarize the data in tabular form.