Introduction to R software for Macs

Sally Eagle

(Updated by Peter J Diggle and Rob Christley)

Version: 09 October 2014

Before we start - Installing R

The R software, and contributed packages, can be accessed and installed by visiting the R project website www.r-project .org.

Here are step-by-step instructions for computers running on a Mac operating system:

1) Go to http://www.r-project.org/
2) Click on CRAN
3) Click on the web-address for one of the 'CRAN MIRRORS' (not critical which but recommended to choose from a nearby country – eg UK or Ireland)
4) Inside the box 'Download and install R' click on Mac
5) Click on R-3.1.1-mavericks.pkg (or the appropriate other version if you are using a different Mac OS operating system, such as Snow Leopard)
6) Once the file has downloaded, double click the file (it is located in your downloads folder)
7) Once the install has run you can open R (R 3.1.1) through the icon in your Applications folder. You can keep the icon in your dock of on your desktop if you want easier access.

*But note that by the time you read this, the version number may have changed

Use of R for simple arithmetic

When you launch R, the program begins with the 'R console' window open. You can type commands directly into this box. You will see the '>' symbol which is basically your prompt to start typing! You need to see this symbol before each command.

R can be used as a simple calculator

You can simply type in commands and press the *return* key to perform simple calculations:

```
20+69                          (type in '20 + 69' after the '>')
[1] 89                         (this is what you should see as your answer)
5*6
[1] 30

(1/5 + 3/10) * 6^3
[1] 108                        (make sure you understand this calculation)
```

Saving your R code

Typing commands directly into R is fine for simple calculations, but for anything complicated it's advisable to save the code so that you can edit it if it contains any mistakes…or if you want to modify it for any other reason.

R for Mac has a built in text editor that can be used for writing and saving your code:

Go to File -> new document (shortcut: command+N) and then save by File -> save as and save the new editor as Intro.R* in a sensible location. Save as an R file (*.R). This editor window now appears with the name Intro.R – R Editor.

*You can use whatever name you like, but mnemonic names are always a good idea. I would also suggest that you set up a new folder for the work you do in the course.

Typing into the editor window:

- When you type commands into this *editor* window you **do not need** to type the '>' symbol.
- When you want to run a command you need to click on the line (if it's just one line of code) or highlight the whole section and then press command+enter.
- Each time you run a new command you need to see the '>' symbol in the console window. If you see a '+' then R is expecting something more from the previous command!

Note that the code you type into the editor should not include the '>' prompt in front of each line. In the remainder of this note, we use italic script to indicate R commands, ordinary script to indicate messages that the system writes to the console

Vectors/Matrices

A vector is simply a sequence of 'elements', typically ordinary numbers. We can define a vector in R using the c() command (c for concatenate).

In the following examples, note that R treats anything you type after a `#' symbol as a *comment.* Including comments in your R code is highly recommended, as it helps you keep track of exactly what your code is doing.

Try the following:

```
vec <- c(1,2,3,4,5)
vec
```

The '<-' (less than followed by minus, commonly called 'gets') symbol has been used to assign what follows to the name given. So here vec 'gets' the vector [1,2,3,4,5]. Typing the name of any object in R simply prints the contents of the object to the console. If you want to save the result of any command, you have to assign it to a named object using the '<-' symbol. Note that the equals symbol (i.e. =) can be used, although for various reasons we prefer to use <-.

We can also set up vectors using functions such as 'seq' (sequence) and 'rep' (repeat):

evens <- seq(0,20, 2) # even.seq 'gets' the even numbers from 0 to 20 in jumps of two

evens
[1] 0 2 4 6 8 10 12 14 16 18 20

ones <- rep(1,20) #repeat the number 1, twenty times
ones # gives the following output:
[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Note the structure of the R code in the above examples; functions (such as *seq* and *rep*) are always followed by other information in round brackets, i.e. (), which provides the information needed for that function to run. We will introduce new functions throughout the course.

Vector calculations

We will use the vector 'vec' we defined earlier to perform some simple calculations on vector elements. For example:

*vec*2*
[1] 2 4 6 8 10

Try the following, noting that you can put several R commands on the same line if you wish, separating them by the semi-colon character

vec+ vec; vec^2; vec+3; sqrt(vec)

Notice how the calculation works on each element of the vector. If, as in one of the above examples, you add a single number to a vector, the single number will be added to each element of the vector. If by mistake you try to add two vectors of different lengths, R will not give you an error message, but it will give you a warning. Try the following:

vec+c(2,4,6,8)

Simple functions on vectors

You can perform simple functions on each element of a vector such as sqrt(vec) and exp(vec) for the square root and for the exponential, respectively. Note that you can also directly perform any of these calculations on a vector without defining it first, for example:

sqrt(c(1,2,3,4,5))
[1] 1.000000 1.414214 1.732051 2.000000 2.236068

We will now illustrate some simple statistical functions.

Set up a new vector called 'age' with the following data (ages of 8 subjects): 21, 35, 18, 22, 24, 33, 37, 41.

age<-c(21, 35, 18, 22, 24, 33, 37, 41)

We can now easily find the mean, variance, maximum, range and so on:

mean(age)
[1] 28.875

var(age)
[1] 74.125

max(age)
[1] 41

range(age)
[1] 18 41

Other useful functions include:

Min() – gives the minimum value

sum() – sum of the elements

sort() - sorts the elements in ascending order

length() – counts the number of elements

summary() – gives the range, interquartile range, mean and median of the data

Note that you can also assign a name to the output to any of these commands if you want to use them later (and it often makes it easier to understand your code when you revisit or if someone else reads it). For example:

mean.age <- *mean(age)* ; mean.age #this does not look like a short cut here but it does help
#when you get to more complicated coding!!

[1] 28.875

Exercises

1) For the following data find $\sum x^2$ (the sum of all elements, each squared)

   3, 7, 2, 6, 1, 3 answer = 108

2) Set up a vector called 'my.seq' which repeats the sequence 0 ,5, 10, 15,...,100 six times, i.e. 0, 5, 10, 15,...,100, 0, 5, 1,0 15,...etc (hint: you need to use the functions seq() and rep() which were used earlier). Check your vector by running :

my.seq

3) The command

x<-runif(n)
generates a vector of n random numbers.

a) Generate 10 random numbers and find the 3rd biggest
b) Generate 10000 random numbers and find the 143rd biggest

Reading in data files

We will now read in an external data set called 'gravity' which holds data (distance and time) from a simple undergraduate physics experiment (covered in the lectures). Data can be read in from various formats such as .csv, .txt, .data, .dbf. Here we have a .data file. For import of other formats see the help pages http://cran.r-project.org/doc/manuals/r-release/R-data.html. Reading in data can be a pain unless you understand exactly how your computer's operating system e.g. (Windows, Mac or Linux) organises its file-structure. If you are stuck try a google search as you can often find help on discussion forums or other websites. However, the following tips should avoid most of these problems:

1. Store your data as a regular array of rows and columns, as a .csv file (columns separated by a comma and, optionally, any number of spaces) and using the two-character string NA (which must be in upper case) to indicate a missing value.
2. Make sure the first line of your data-file contains mnemonic names for each of the columns, again separate by commas (example below)
3. Read the data-file with the R command
   data<-read.csv(file.choose())

This will open a window from which you can browse through your folders until you find the data-file you want. Opening it will then cause its contents to be read into R.

Example. Create a file called "testdata.csv" in a folder called "statscourse" containing the following information:

x,y
1, 5.1
2, 6.3
3, 7,7
4, 8.5
5,10.1

You can do this either using a text-editor or entering the information into an Excel spread-sheet with two columns and saving the result as a .csv file.

Within R, try the following:

data<-read.csv(file.choose)) # navigate to "statscourse" find "testdata.csv" and click to open
names(data)
dim(data)
plot(data$x,data$y) # data$x means "the column named x in the file called data"

Once you are comfortable with this, you can find out about other, more flexible ways of reading data-files in various formats. One that you will need for the exercises is the following.

data<-read.table(file.choose(),header=TRUE)

This operates in the same way as the previous example, except that it expects the data to be in a .txt file with columns separated by one or more spaces (rather than commas), and you have to tell the function that the first line of the file contains the names of the columns (also separated by one or more spaces).

Exercise

Visit the web-page

http://www.lancaster.ac.uk/staff/diggle/intro-stats-book/datasets/

go to the line "Crossover trial of asthma treatments" and click on description. After you have read this, click on data and download its contents. Read the data into R and make a scatterplot* of each child's F result against their S result. Which of the two drugs do you think is the more effective in treating an asthma attack?

*a scatterplot of pairs of values (x,y) is a plot in which each pair is shown as a dot at position x on the x-axis and y on the y-axis. To do this you will need to use the *plot()* function.

Simple graphics

We have already seen a simple example of a scatterplot. If x and y are vectors of the same length, a scatterplot of x against y is obtained from the command

plot (x,y) #if you don't have objects called x and y you will get an error

You can customise your graphs in various ways, and superimpose pots. Try the following:

```
x<-1:10                          # 1:10 is a convenient shorthand for c(1,2,3,4,5,6,7,8,9,10)
y<-c(1.3,2.5,3.5,4.2,5.9,7.0,7.8,8.5,9.9,11.1)
plot(x,y,type="l")
plot(x,y,pch=19,col="red")
plot(x,y,cex=2);   lines(x,y); points(5,5,cex=3,col="red",pch=19)
```

Plots can also be saved as pdf files so that they can be used in other documents.

```
par(pty="s")                                    #this gives a square plot window
pdf("scatterplot.pdf",height=6,width=8)         # height and width are in inches
plot(x,y, pch=19, xlim=c(0,10),ylim=c(0,12),xlab="this is x",ylab="and this is y", col = "red")
dev.off()
```

The 'scatterplot.pdf' file now appears in your folder and you should be able to open it as a pdf document.

Exercise

Notice the other inputs (arguments) to the plot function which are used for improving the appearance of the plot. The arguments xlim/ylim set the limits of the x-axis/y-axis. Those containing 'cex' set the text size (axis and labels), 'pch' sets the symbol used for plotting and 'col' sets the colour of the plotted points or lines. Experiment with changing some of these inputs.

Help pages

If you want to find out the inputs needed/available for a particular function, what the function does and how the result is stored you can access the details by simply typing the following :

?plot

or

help(plot) #for the plot function

Exercises

1) Using the plot function help page add a title to your scatterplot.pdf file

2) Now save and read in a new dataset - blood pressure data - from the website www.lancs.ac.uk/staff/diggle/intro-stats-book/datasets in the same way that we read in the asthma data. This data gives synthetic BMI and blood pressure data for 30 study patients, 15 of whom were assigned to receive drug A and 15 assigned to drug B. Have a look at the data to see how it is laid out and make sure that the column names are defined for use later.

   Now we can produce a plot of BMI vs blood pressure indicating which patients received A and which received B

3) Run the following code line-by-line, filling in the gaps indicated by ???. Check you understand what is happening in each step as this is a little more complicated than the earlier plots. The last line gives an alternative way of saving plots to pdf files.

```
plot(c(BMIA,BMIB), c(BPA,BPB), type="n", xlab="BMI", ylab="BP", cex.lab=1.5, cex.axis=1.5,
xlim=c(22,30), ylim=c(???,???))          #this produces the plot region and labels
points(BMIA, BPA, pch=19, cex=1.5)       #this adds the points for the drug A patients
```

points(BMIB, ???, pch=1, cex=1.5,col="red") #this adds the points for the drug B patients
title("???") #whatever you like
dev.print(device=pdf, file="BP_BMI.pdf", width = 8, height = 6)

There are some 'optional extras' below that you can complete now or in your own time. Otherwise you can now end the R session.

You can close R by simply closing all the windows, but the recommended way is to type the command

q() # for "quit"

You will be asked if you wish to save the workspace. This is useful if you want to save the objects you have created in the session.