August 29, 2012

Statistical Data Analysis: R Tutorial

Dr A. J. Bevan*,

Contents

1 Getting started

R is an open source statistical programming language which can be downloaded from the following website:

$$http://www.r-project.org/$$

This tool may be useful for this course, and some sectors value familiarity with R. You may wish to use these tutorials to introduce yourself to this package to bolster your CV. In these notes type in bold is entered into a command line terminal, and itallicised text appears on the command line as a result.

In order to start R from a computer that has this programme already installed type

R

This should result in the following output being displayed on the command line terminal

*a.j.bevan at qmul.ac.uk

R version 2.15.0 (2012-03-30)
Copyright (C) 2012 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: x86_64-apple-darwin9.8.0/x86_64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

In order to quit an R session one simply enters **q()** at the prompt. More information on the use of R can be found online at the R project website and as the result of performing a google search for 'R'. In addition to this there are a number of books available on the use of this package.

2 Basic calculations

You can perform simple arithmetic in R in a logical way using operators akin to any other programming language. For example

3*5
[1] 15

3/2
[1] 1.5

7-2+1
[1] 6

In addition to this one can define arrays of numbers and perform simple calculations on these sets of data. The following example creates an ordered list of data in an array with variable name **x**, and computes the arithmetic mean, variance and standard deviation for these data.

x <- c(1,2,3,4,5,6)

print(x)
[1] 1 2 3 4 5 6

mean(x)
[1] 3.5

var(x)
[1] 3.5

sd(x)
[1] 1.870829

There is a command line help function that enables you to find out more about a particular R command. For example if you wish to query the help for the sd function simply enter **help(sd)** at the R prompt. This will result in the following output.

Standard Deviation

Description:

This function computes the standard deviation of the values in x. If na.rm is TRUE then missing values are removed before computation proceeds.

Usage:

sd(x, na.rm = FALSE)
...

Exercises

1. Define the variables $\theta = 3.2$ and $\phi = 1.7$ and print out these variables.
2. Compute $\theta + \phi$.
3. Compute $\theta - \phi$.
4. Compute $\theta\phi$.
5. Compute θ/ϕ.
6. Define an array x containing the prime numbers between 1 and 20.
7. Print out the array x.
8. Compute the arithmetic mean, variance and standard deviation of x.

3 More advanced calulations

It is possible to perform more advanced calculations using additional functionality in R. For example if you wanted to compute the standard deviation without using the in-buit function **sd(x)**, for some data x you can do so via the following steps

```
x=c(0.5,0.9,1.2,1.5,1.8,2.0,3.4,4.1,5.0,5.1,7.5,8.5)
meanX = mean(x)
stdevX = sqrt( sum( (x - meanX)^ 2 ) / (sum( !is.na( x ) ) -1 ))
```

The first line defines the data sample, the second comutes the arithmetic mean of the data using the **mean** function, and the third line computes the value of the sum over all $(x_i - \overline{x})^2$ terms and normalises that by $N - 1$. The term **!is.na(x)** is a boolean check that the value of each data point is defined as a number, as opposed to NaN (Not a Number), so that the denominator is simply the sum over the number of valid data, less one (to include Bessel's correction for an unbiased variance).

Several features of R have been introduced in this example

- Operations on sets of data stored in an array without the need to iterate through the elements. This can be seen with the term **x-meanX**, where the scalar value **meanX** is subtracted from each element in **x**.

- The ability to sum over an array of data is computed using the **sum** function.
- One can check if a number is well defined using **is.na(x)**.

If you've not encountered NaN before in computing, the following provides a simple illustration:

y = 0/0
print(y)
[1] NaN
is.na(y)
[1] TRUE

A number is NaN if it is ill defined such as the ratio of some number divided by zero.

This brief introduction should provide you with a good starting point for using R to analyse data. There is a lot more that you can do with this programme, and some additional information that you may find useful for project work are introduced in the rest of this brief note.

If you want to learn more about how to use R you are advised to look online or check the library for books on this topic.

Exercises

1. Using the data set $\Omega = \{1, 1.2, 1.5, 1.7, 2.3\}$ compute the arithmetic mean without resorting to the built in functions within R.
2. Using the data set $\Omega = \{1, 1.2, 1.5, 1.7, 2.3\}$ compute the standard deviation without resorting to the built in functions within R.
3. Using the data set $\Omega = \{1, 1.2, 1.5, 1.7, 2.3\}$ compute the skew without resorting to the built in functions within R.

4 Plotting data

The **plot** command can be used to plot data - however one should always strive to produce a meaningful figure with appropriate axis labels so that it is obvious as to what has been plotted. Given some data stored in an array one can simply plot the data as follows

x <− c(1,2,3,4,5,6)
plot(x)

This will produce a simple graph with the index number along the horizontal axis and the data corresponding to the value of each index plotted on the vertical axis. Having produced such a plot the next step is to be able to format the axis labels and title of the plot so that is it suitable for external consumption[†]. At the time of plotting one can specify a number of options including main and sub-titles as well as axis labels. For example

x <− c(1,2,3,4,5,6)
plot(x, main="Main Title", sub="A sub title", xlab="x axis label", ylab="y axis label")

will produce a plot like the one shown in Figure 1.

It is possible to print out pdf files in R by opening a file (by default if no filename is specified then a file called *Rplots.pdf* will be created. Having opened the pdf file one can plot data, and finally close the file. The closure of file is done using the **dev.off()** command. The following example creates a pdf file of the plot shown in Figure 1.

[†]i.e. suitable for someone other than the creator of the plot to view.

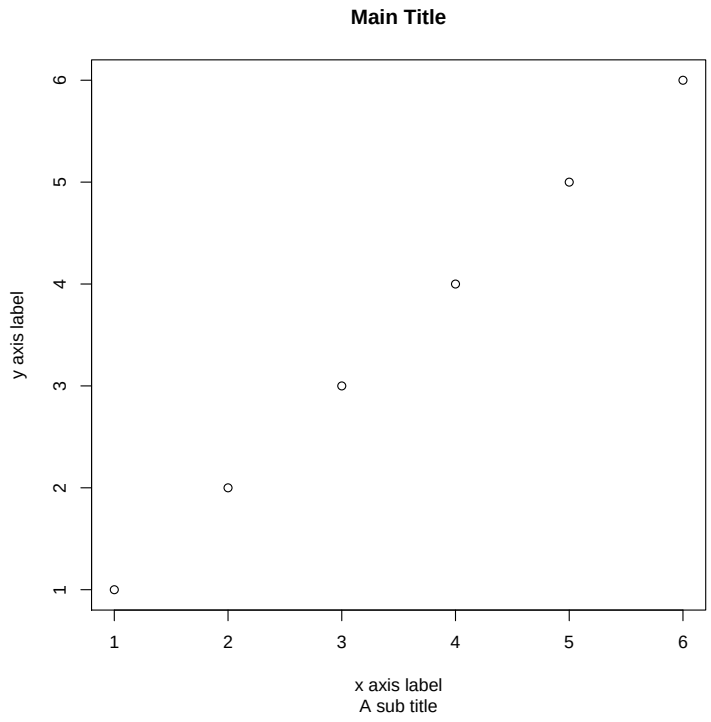

Figure 1: An example plot created using the R commands given in the text.

pdf()
plot(x, main="Main Title", sub="A sub title", xlab="x axis label", ylab="y axis label")
dev.off()

Exercises

1. Create a plot of the following data: $\Omega = \{1, 1.2, 1.5, 1.7, 2.3\}$ as a function of the index of each data point.
2. Add axis labels to the plot you've created.
3. Print the plot to a file - try giving this file a name (pass a string argument to the **pdf** command).

5 The quantmod package

The quantmod package is a "Quantitative Financial Modelling & Trading Framework for R". This package is described in full on the package web page

$$http://www.quantmod.com/$$

First in order to use quantmod the package has to be installed. This can be done using

install.packages("quantmod")

Once installed it is possible to download share prices for further manipulation. For example if you wanted to view the share price of LLoyds, the trading symbol on the FTSE100 is LLOY.L and one can download

share prices into a local variable called **LLOY.L** and compute the return on this stock as the local variable **LLOYDS** using the following code snippet.

```
require(quantmod)
setDefaults(getSymbols, src='yahoo')
getSymbols('LLOY.L')
plot(LLOY.L)
LLOYDS <- dailyReturn(LLOY.L)
plot(LLOYDS)
```

The function **plot** can be used to plot the data on screen as was the case for the share price and the daily return above. Figure 2 shows the distributions of share price and daily return plotted for the requested period (back to January 2007 when the yahoo data became available).

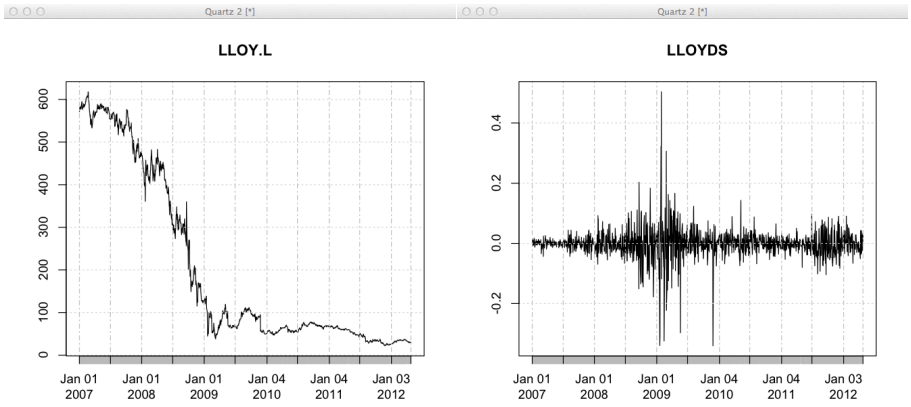

Figure 2: The share price (left) and daily return (right) for Lloyds TSB. The data were extracted using the quantmod package.

The quantmod package has a lot more functionality than this simple illustration and anyone interested in learning more should refer to the documentation and examples found on the web.

More information (including extensive source of documentation) can be found on the R webpage.