Getting Started in R

Part 1

by M. Papathomas and E. Rexstad (with material from a document prepared by R. King)

```
## Warning: package 'knitr' was built under R version 3.2.5
```

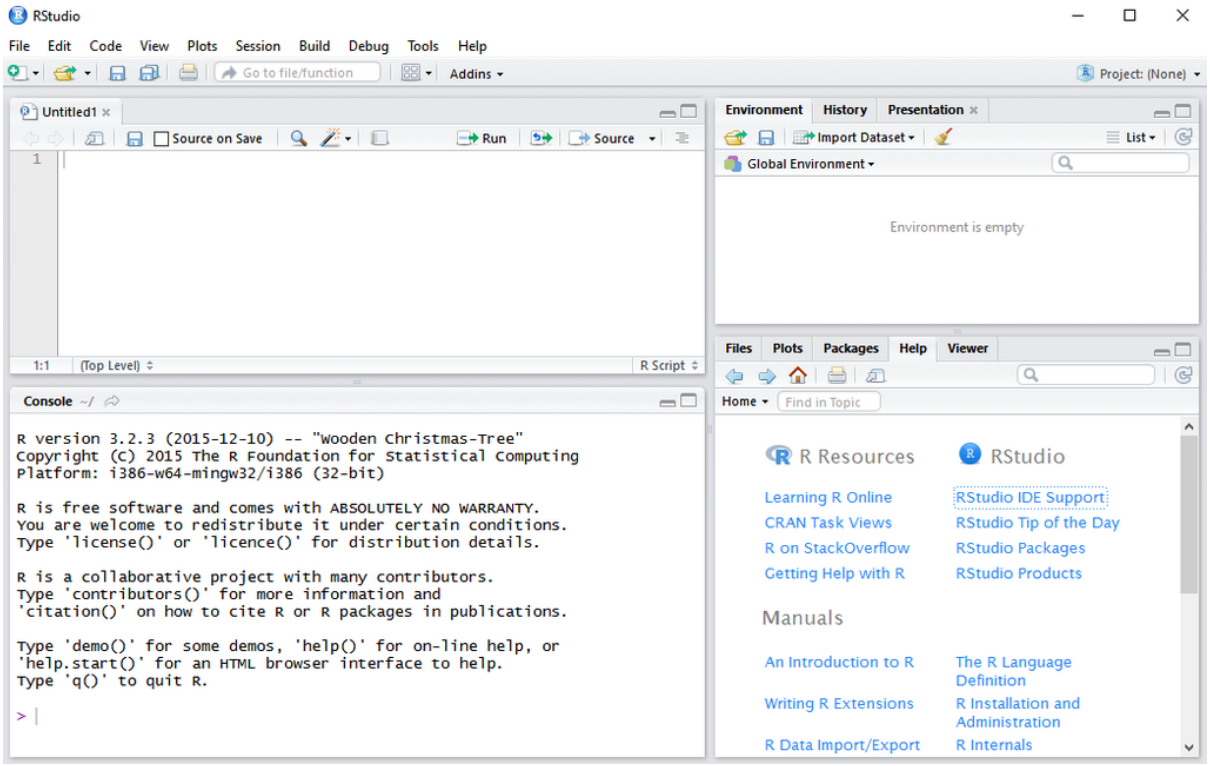
Use the first practical session to become familiar with the basic operations in R, how to access it, use the help and build up the knowledge of simple (and more complex) commands. However, do not treat the preliminaries as a rapid typing exercise; make sure that you understand each step before proceeding to the next. You will be using R extensively for the assessed practical, which accounts for 15% of your total mark. If you have not used R before, you should work through the following tasks carefully. If you have used R before, you may find these tasks useful revision - and don't forget the help functions within R.

There are many resources on the Internet to help learn R. A fine book entitled *The Art of R Programming* by Norman Matloff is available electronically via the Library. Some of the useful sections of this book are,

Section	Topic
1.2	First R session
1.4	Data types
1.7	How to get help
2.1	Vectors, arrays and matrices
2.4	Vector operations
5.2	Data frames

Starting R-Studio

The most useful interface to R is R-Studio. There is plenty of online help for using R-Studio (for example, this link). This document will describe some statistical computations with R, carried out using R-Studio.



Examine the screen shot above, showing the layout of the R-Studio development environment. R-Studio is opened by choosing R-Studio from the `Start menu` of Windows 7 as found in the MicroLab.

- Lower left panel is the `R console`; this is where you could directly submit commands to the R language interpreter. Most of the instructions provided in this handout can be carried out using this panel. As you begin to create more sophisticated sets of instructions, you will find using the `editor` panel to be more convenient.
- Upper left panel is the `editor` where you can type a series of commands. Nothing happens when commands are typed into this `editor`, but the commands need to be sent from the `editor` panel to the `console` panel. This can be done by highlighting lines of `R` code in the `editor` and pressing the `Run` button on the right of the menu line inside the `editor` panel. Alternatively, you can use the `<ctrl>-Enter` keyboard shortcut (Enter key pressed while holding the `ctrl` key).
- Upper right panel usually shows the `environment` which is the objects in the R `workspace`. You will hear more about `objects` shortly.
- Lower right panel shown in the figure is a way to access help regarding use of `R` or use of `R-Studio`. This panel contains tabs at the top of the panel indicating that in addition to accessing `help`, this panel can also be used to view
  - `Files` on the computer,
  - `Plots` created by your `R` code,
  - `Packages` available for use with your code.

Help Commands

These are one of the most helpful (and widely used) functions used in `R`. The `help()` function in `R` provides information on how to use the functions in `R` including necessary input parameters, information relating to the statistical functions themselves, related commands and often examples. The help can be initiated either via the lower right panel or via the command-line in the Console window.

For example, if you know the `R` functions `mean` that you are interested in and want to find out more, try typing:

```
help("mean")
```

in the command line, or simply `?mean`. This will display help about the `mean()` function in the lower right panel of R-Studio.

Alternatively, if you did not know the command in `R` and wanted to find help on the topic mean, try typing:

```
help.search("mean")
```

This provides a list of commands relating to the topic mean, and the relevant function can be determined. If you cannot easily find the function of interest using R's internal `help()` function, you might also use a Google search for help on topics in R.

Try these on your own

- Find out what the function 'sample(c(0,1,2,3),1,replace = TRUE)' does?
- Use the 'help' function (or Google) to find the function for evaluating the number of ways we can choose r elements (disregarding their order) from a set of n elements, ${}^nC_r$. Use this function to evaluate ${}^{10}C_4$.

Using Arithmetic Commands

Basic arithmetic commands can easily be implemented with `R` to perform calculations. The four most basic commands are `+`, `-`, `*` and `/`. Additionally, the power function is denoted by `^`.

For example, to evaluate $(8^3)/2 - 7$, type the following in the console panel:

```
8^3/2-7
```

This is equivalent to `((8^3)/2)-7`.

Try these on your own

- Evaluate the square root of 8^3 (note - this can be done in more than one way).
- Evaluate $10^{\frac{1}{4}}/5^{\frac{2}{3}}$.
- Calculate 132×87.
- Evaluate $\log_5(10)$.

Setting and Removing Objects

`R` functions operate on objects. The simplest object is a scalar, which may be numeric, or a named variable. For example, the following command creates a variable `b`, and assigns the value 5 to it:

```
b <- 5
```

The assignment operator is `<-`, which might be interpreted as meaning *comes from*, as opposed to the more usual `=` symbol, which will be used to assign values to arguments passed to functions. To check the above command, simply type

```
b
```

`b` in this example is essentially a single-element array. We can create a three-element array, with values 5, 1 and 2, as follows:

```
a <- c(5,1,2)
```

Check this result.

In mathematics, a one-dimensional array with > 1 element is called a vector, and a two-dimensional array a matrix. To create a matrix, use the command `matrix`. For example,

```
a2 <- matrix(c(2, 4, 3, 1, 5, 7), nrow=2, ncol=3)
a2
```

```
##      [,1] [,2] [,3]
## [1,]    2    3    5
## [2,]    4    1    7
```

creates a 2×3 array with elements $(1, 3, 5, 2, 4, 6)$. Note how R arranges those elements in the matrix, in relation to their order in the vector in the 'matrix' command.

You can print out a single element of an array to the console:

```
a[2]
a2[2,3]
```

Change the value of those elements to -3.5.

Now investigate the following commands. Make sure that you check the result, and understand what each has done, before going to the next. While you are entering these `R` commands, note the changing contents of the upper right `environment` panel of R-Studio, showing the changes being made to the objects in the R environment.

Try these on your own

```
a <- 0:100
c <- a[a>49]
a <- a/100
i <- c(1,5,10,20,30)
a[i]
a[50:101]
a[1:4] <- c(3,2,1,0)
a2[2,1:3]<-c(7,8,9)
```

In addition to using the `environment` panel, a list of the objects in the environment can be produced with this function:

```
objects()
```

This allows you to keep track of the objects you have created. To remove the object `a`, use the `rm()` function:

```
rm(a)
```

You can remove as many objects as you want by listing them, separated by commas. Remove all the objects you have created and check that they have gone.

Try these on your own

- Create a vector of length 5 with elements 0, 2, 4, 6, 8 and put this into some object (choose the object name yourself).
- Create a new object with elements equal to 2 times that of each element in the previous object created WITHOUT physically typing in each element.
- Create a vector of length 100, such that the first 50 elements are equal to $0.0, 0.01, 0.02, \ldots, 0.49$; the next 30 elements are equal to $1.0, 1.01, 1.02, \ldots, 1.29$ and the final 20 elements are equal to $20, 19, \ldots, 1$. HINT: you will find the 'seq()' function useful for creating vectors with elements that are sequences of numbers. Explore the function 'seq()'.
- Create a 40×35 matrix with all elements equal to 5. HINT: you will find the command 'rep(5,40*35)' useful; explore the function 'rep()'.
- Remove the objects that you have created.

How to save and execute your code

Sometimes you will be giving commands directly in the Console window on the bottom left. However, if you are writing code that you may want to use again, i.e. almost always, you should be writing the commands in the top left window. Then, you can save all the code you have written as a file with extension .R. To do this, you can simply click on the well known `Save` icon on the top left, or click on `File` and then choose `Save as`. One way to execute the code you have written in the .R file is to select the lines you want to execute and then click on the `Run` icon at the top right of the window.

Distributional Calculations in `R`

Usually, four R commands correspond to a specific distribution. Here, we consider the Binomial distribution as an example. The Binomial distribution gives the probability that we observe t successes after n trials, with the probability for a succesful trial equal to p. The simplest version of the four `R` commands are:

- `dbinom(t,n,p)` ← gives the probability of observing t successes (probability mass function);
- `pbinom(t,n,p)` ← gives the probability of observing t or fewer successes (cumulative distribution function);
- `qbinom(q,n,p)` ← gives the smallest value of t such that the probability of observing t or fewer successes is greater or equal to q (inverse cumulative distribution function);
- `rbinom(m,n,p)` ← creates m simulations. For each simulation it performs n trials, and returns the number of successes for each simulation (generates m random deviates from the given Binomial distribution);

Similar commands are available for other standard distributions. For example, for the Normal distribution we have `dnorm`,`pnorm`,`qnorm`, and `rnorm`. Commands also exist for the Uniform, Exponential, Poisson, F, (Student's) t, χ^2, etc. Note that the arguments for these functions and their interpretation differ for different distributions.

Try these on your own

- Assume that some medicine is successful with probability 0.4, and that 10 patients take this medicine. Calculate the probability that 6 patients will recover.
- Simulate a random deviate from a $U[0, 1]$ distribution.
- Let $X \sim Po(10)$. Calculate $\mathbb{P}(X = 11)$. Now calculate $\mathbb{P}(X \geq 12)$ and $\mathbb{P}(X \geq 11.1)$. Comment on (and explain) the results obtained.

Plotting Graphics

You will now use the graphics window. Suppose we wish to simulate many times the Binomial experiment that looks at 10 patients (trials) with probability of success 0.4, and plot the results in a histogram. You may ask why we want to do this. One of the possible answers is that we may want to visualise the probability that corresponds to a certain number of patients recovering, and see how the different probabilities compare to each other. For example, is the probability that 6 patients recover much smaller compared to the probability that 7 patients recover?

```
hist(rbinom(1000,10,0.4))
```

Note that `rbinom` generates the deviates, and passes them to the `hist()` function. We could do this in two steps, by assigning the deviates to a variable `x` say:

```
x <- rbinom(1000,10,0.4)
hist(x)
```

To compare plots in the same window, use the following sequence of commands:

```
par(mfrow=c(2,2))
hist(rbinom(30,10,0.4))
hist(rbinom(100,10,0.4))
hist(rbinom(300,10,0.4))
hist(rbinom(1000,10,0.4))
```

The `par()` function has split the window into (in this example) 2 rows and 2 columns. To return to a single plot in the graphics window, type `par(mfrow=c(1,1))`. If you are using the `Console` window, you do not have to type the `rbinom()` function in full four times; you can use the up arrow to recall the command `x <- rbinom(30,10,0.4)`, and edit *30* to *100*, by using the left arrow key and editing as required.

Now, suppose that we wish to plot the vector `x` that we created above. Then, to plot `x[i]` against `i` type:

```
plot(x)
```

Note that this is a Scatter plot and not a Histogram. You can customise the plot by adding labels and specifying that you want a solid line to join the points, rather than asterisks at each data point. Type

```
plot(x,type="l",xlab="i",ylab="x[i]", main="Plot of deviates from Bin(10,0.4)")
```

To see what other options are available, use the `help()` function.

Try these on your own

- Read the following time series data into a vector:

$$10, 12, 16, 14, 19, 17, 13, 21, 18, 15, 19, 25, 27, 20, 14, 12, 17.$$

  - Plot a histogram of this data.
  - Plot this data using a line graph, such that the limits of the y-axis are [0,30]. (Explore the 'plot' function to find how to control the limits in the two axes.) Give the graph a title.

- Suppose that we observe the paired data (each data point in line one (vector a) is associated with the corresponding data point in line two (vector b)):

a	453	312	514	407	612	198	276	477	501	412
b	12	9	15	10	14	9	11	13	14	12

Plot a scatter graph of the paired data of vector a against vector b, including relevant axis labels and a title. HINT: try 'plot(a,b)' adding arguments to the function 'plot' to make the graph more informative.

- We observe the following time-series data ($t \equiv$ time):

t	0	1	2	3	4	5	6	7	8	9	10
y	150	147	155	167	157	159	143	150	165	160	153

Plot the data points with time along the x-axis. Next *join-the-dots* on this graph to produce a line-graph (but with the plotted points still present). Find a function that allows us to do this and use it to produce the desired graph.