

R statistical computing laboratory practicals

Paul S.P. Cowpertwait
IIMS Massey University Albany

Preliminary Note

These practicals and exercises are appropriate for Level One Statistics (e.g. 161.100/120/130). Remember that the best way to learn statistics and programming with data is by doing the exercises yourself. For these exercises, you need to be at a computer that has the R software installed (see Section 1.1 to install R on your own computer if you are doing the exercises at home).

Text Book References

MM – Moore and McCabe, 3rd Edition of ‘Introduction to the Practice of Statistics’;
HK – Higgins and Keller-McNulty, 1st Edition of ‘Concepts in Probability and Stochastic Modelling’; VR – Venables and Ripley, ‘Modern Applied Statistics with S’

1 Introduction to R and S Programming

R is a computing environment which incorporates the *S programming language* originally developed by Chambers et al at the Bell Laboratory (where the C programming language was also developed). The S language was specifically designed for flexible programming and analysis of data, and consequently has become a popular tool with statisticians, programmers, and data analysts. The language is similar to the commercial package ‘S-Plus’, although R is a more efficient implementation in terms of computing speed.

1.1 Downloading R for personal use

R can be downloaded free of charge via the web site:

<http://cran.r-project.org/>

If you decide to download R for your personal use, then first go to the pre-compiled (binary) links at the above site and follow the instructions for ‘Windows’ or ‘Linux’ (depending on the operating system on your computer – there is probably an alternative for Mac users which I haven’t investigated yet). You need to choose ‘base’ followed by ‘setup’ and follow the instructions given.

R comes with no warranty or licensed support (because it is free). However, you can subscribe to an active mail base for asking questions or seeking advice on problems related

to R. To do this, send an email to `Majordomo@stat.math.ethz.ch` with the command: `subscribe r-help` in the message.

1.2 Getting Help

Start R (in Windows select the executable program from the ‘start’ menu; in Linux type ‘R’ at the command prompt). At any time you can quit by typing `q()` (you will be asked whether you want to save the ‘workspace image’ – in general say ‘no’). To get help on any command type `?name`, where `name` is the name of the command you need help with. The only command you know so far is `q()` so try typing: `?q`. After you’ve looked at the help on ‘q’ close the help window.

You can scroll through all commands that you’ve used by using the ‘up’ and ‘down’ arrows on your keyboard. This can be useful if you want to execute the same command again or if you want to minimize typing by ‘editing’ a previous command. Try pressing the ‘up’ and ‘down’ arrows to see what happens.

You can also get help via a hyperlink html system. From Windows click on ‘help’ on the console menu bar, followed by ‘help html’; in Linux (or Windows) type `help.start()` at the R command prompt. You will get a range of html links to choose from. Some useful ones for you to try are:

- ‘An Introduction to R’ – a good introductory text;
- ‘Packages’ – a list of libraries that can be loaded into R with the `library()` command. The ‘base’ package has most functions you will need in it. Click on ‘packages’ followed by ‘base’ to get an alphabetical listing of available functions in the base package;
- ‘Search Engine’ – useful if you know what you want to do, but don’t know the right command or function (try typing ‘boxplot’ in the search engine to see what it returns).

1.3 Entering Data

There are three main ways of getting data into R:

1. Typing the data in using the keyboard – function `c()`;
2. Reading in a single variable, in no particular format – function `scan()`;
3. Reading in a table of one or more variables – `read.table()`.

As an example of 1 above, create a vector of the amount spent in a supermarket (MM, p12) by the first 4 ordered customers:

```
> spent <- c(3.11, 8.88, 9.26, 10.81)
```

Notes:

- An assignment in R is written as `<-`.
- The function `c()` stands for ‘concatenate’ and places the data into the vector ‘spent’.
- Comments in R are preceded by the hash symbol: `#` (everything on a line after a `#` is ignored)

Type `spent` to view the elements of the vector (anything after `#` is ignored):

```
> spent # note: this comment does not affect the command!  
[1] 3.11 8.88 9.26 10.81
```

Note you can access the elements of a vector by referring to its index in *square* brackets, e.g.

```
> spent[3]  
[1] 9.26
```

gives the amount spent by the third ordered customer. Note: this is useful because it means we do not have to scroll through copious output to get the information we want.

As an example of 2 above, we will scan unsorted data (MM Example 1.5) into the vector ‘spent’ and then sort the data into ascending order. The data are in a file called ‘spent.dat’ on the H-drive (or at <http://www.massey.ac.nz/~pscower/161330> – if you use the web site you will need to save the data to a file before reading it in). The data can be read using ‘scan’ with ‘file.choose()’ to open a window for selecting the data file, or you can just type in the name of the data file in quotation marks: ‘/path/spent.dat’ (you will need to enter the path where the data has been saved). Read the data into R using the ‘scan’ command:

```
# ‘file.choose’ creates a window for selecting files  
# data will be stored in the vector ‘spent’  
> spent <- scan(file.choose())  
Read 50 items  
  
# list the 50 observations:  
> spent  
[1] 17.39 39.16 42.97 41.02 28.38 19.50 10.81 19.27 9.26 25.13 20.16 18.36  
[13] 24.47 93.34 48.65 19.54 59.07 17.00 32.03 52.75 22.22 44.67 18.43 27.65  
[25] 28.06 26.26 13.78 54.80 61.22 46.69 8.88 34.98 23.04 15.62 15.23 50.39  
[37] 86.37 12.69 36.37 45.40 85.76 26.24 24.58 44.08 70.32 3.11 20.59 28.08  
[49] 82.70 38.64  
  
# These observations are unordered. Put them into ascending order:  
  
> spent <- sort(spent)
```

```
> spent
 [1]  3.11  8.88  9.26 10.81 12.69 13.78 15.23 15.62 17.00 17.39 18.36 18.43
[13] 19.27 19.50 19.54 20.16 20.59 22.22 23.04 24.47 24.58 25.13 26.24 26.26
[25] 27.65 28.06 28.08 28.38 32.03 34.98 36.37 38.64 39.16 41.02 42.97 44.08
[37] 44.67 45.40 46.69 48.65 50.39 52.75 54.80 59.07 61.22 70.32 82.70 85.76
[49] 86.37 93.34
```

```
# Note: the same output could be obtained by typing 'sort(spent)'
# but this would leave the vector spent unchanged (i.e. unsorted)
```

```
# The vector 'spent' now contains the data given in MM, Example 1.5.
```

1.4 Basic Statistical Functions

Once the data is in a vector, there are plenty of functions for finding useful summary statistics, e.g. the mean, median, standard deviation, variance, and quantiles. Work through the examples below, checking your output against mine. Remember that comments are given after the hash (#) symbol (and you don't need to type them in!).

```
> mean(spent)
 [1] 34.7022
> sd(spent)
 [1] 21.6974
> var(spent)
 [1] 470.7771
> median(spent)
 [1] 27.855
> quantile(spent, 0.5) # this, of course, is the same as the median
 50%
27.855
># note for the next one use 'up' arrow and edit the '0.5' to '0.25'
> quantile(spent, 0.25) # the lower quartile = 0.25 quantile = 25th percentile
 25%
19.3275
># upper quartile:
> quantile(spent, 0.75)
 75%
45.2175
> IQR(spent) # note the capitals: R is CASE sensitive!
 [1] 25.89
> quantile(spent, 0.75) - quantile(spent, 0.25)
 75%
25.89
> max(spent)
 [1] 93.34
> min(spent)
```

```
[1] 3.11
># The following gives the 5 point summary and the mean:
> summary(spent)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.11  19.33   27.85   34.70  45.22   93.34
> fivenum(spent) # five number summary
```

Look at the details for one or more of the above commands using ?name.

1.5 Basic Univariate Plotting Functions

Work through the following. Type in the commands and read the comments after #.

```
# Our first two commands reproduce Fig 1.3, in MM, p12.
> stem(spent)
```

The decimal point is 1 digit(s) to the right of the |

```
0 | 399
1 | 1345677889
2 | 000123455668888
3 | 25699
4 | 1345579
5 | 0359
6 | 1
7 | 0
8 | 366
9 | 3
```

```
# Using the 'scale' parameter the stems can be split as follows
> stem(spent, scale=2) #'doubles' the scale
```

The decimal point is 1 digit(s) to the right of the |

```
0 | 3
0 | 99
1 | 134
1 | 5677889
2 | 0001234
2 | 55668888
3 | 2
3 | 5699
4 | 134
4 | 5579
5 | 03
5 | 59
6 | 1
```

```
6 |  
7 | 0  
7 |  
8 | 3  
8 | 66  
9 | 3
```

```
> hist(spent) # a graphics window should open with the picture  
  
# the number of 'bins' can be changed using the 'break' parameter  
> hist(spent, br=10)  
> boxplot(spent) # see MM Fig 1.15, p49  
> qqnorm(spent) # see MM Fig 1.32, p83
```

1.6 Normal Quantiles and Exercises from MM

The functions `qnorm` and `pnorm` can be used to find normal quantiles corresponding to particular probabilities and probabilities corresponding to particular quantiles (so we can use R like statistical tables, but with greater accuracy). We can use these functions to answer Q1.81, Q1.82, and Q1.87 in MM, p88, 89. Here is the solution in R:

```
# MM Q1.81(a)  
> pnorm(2.85)  
[1] 0.997814  
  
# MM Q1.81(b)  
> 1 - pnorm(2.85)  
[1] 0.002185961  
  
# MM Q1.81(c)  
> 1 - pnorm(-1.66)  
[1] 0.9515428  
  
# MM Q1.81(d)  
> pnorm(2.85) - pnorm(-1.66)  
[1] 0.9493568  
  
# MM Q1.83(a) - have a go at working the rest out on your own before looking at solution  
> qnorm(0.25)  
[1] -0.6744898  
  
# MM Q1.83(b)  
> qnorm(0.6)  
[1] 0.2533471  
  
# MM Q1.87(a)  
> 1 - pnorm(700, mean=544, sd=103)
```

```
[1] 0.06494154
```

```
# MM Q1.87(b)
```

```
> pnorm(500, mean=544, sd=103)
```

```
[1] 0.3346225
```

```
# MM Q1.87(c)
```

```
> pnorm(800, mean=544, sd=103) - pnorm(500, mean=544, sd=103)
```

```
[1] 0.6589079
```

```
>
```

Work through the above solution checking that you understand each step. Then attempt Q1.93 and Q1.94 in MM, p90.