MATH1019 Linear Algebra and Statistics for Engineers

Laboratory 2

1 Introduction

In the last lab, we learned some simple commands in *R* to carry out basic arithmetic operations, to create vectors, to apply functions to scalar and vector quantities, and to plot some simple graphs. In this lab, we will learn some additional *R* commands as well some additional plotting functions that will be useful for exploratory data analysis.

2 Reading and Saving Data in R

R has the ability to read data from external files stored in several formats. For all but the smallest of data sets, when working with data stored in a format not readable by *R*, it will almost always prove easier first to save the original data as a text file and then to read the external file using read.table() or scan(). Typically read.table() is more user friendly, although scan() reads large data of a single mode more quickly than does read.table(). It also reads data from the console. To download a wide variety of files from the Internet, one can use the function download.file(), which allows the user to specify where they want the downloaded file saved by using the destfile= argument.

2.1 Using read.table()

The function read.table() reads a file in table format (a rectangular data set) and creates a data frame from that file. The file may be on your computer, at an unsecure (http) website or, for Windows users, the contents of the clipboard (file = "clipboard"). To create an R data frame from an external file, one can use the function read.table() or one of its variants. Consider the text file FAT.txt available online from the website:

http://www.appstate.edu/~arnholta/PASWR/CD/data/Bodyfat

```
> site<-"http://www.appstate.edu/~arnholta/PASWR/CD/data/Bodyfat"
> FAT<-read.table(file=site, header=TRUE, sep="\t")
> head(FAT)
  age  fat sex
1  23  9.5   M
2  23 27.9   F
3  27  7.8   M
4  27 17.8   M
5  39 31.4   F
6  41 25.9   F
```

The website's address is stored in the R object site. This is done so that the read.table() command can appear on a single line. Inside the function read.table(), the only mandatory argument is the file location. Since Bodyfat.txt has column names and is a tab-delimited file, the arguments header = TRUE and sep = "\t" are used. There are numerous arguments for read.table(): the field separator character (sep=), which by default is set to a blank space; the character used for decimal points (dec=); the character vector used to represent missing values (na.strings=); and many others. To read about all of the arguments for read.table() and its

variants such as read.csv(), which can be used to read comma-separated value (.csv) files, type ?read.table at the R prompt. If the Bodyfat.txt file were stored in the current working directory of R, then one would only need to specify the file name in quotes of the file= argument. When files are not in the working directory, then the complete path to the desired file must be used.

To select a file from the windows file menu simply type the argument file.choose(), as in: raed.table(file.choose(), header = TRUE). This will open the windows file explorer menu and a file can be chosen from an appropriate directory.

2.2 Saving Data Frames to External Files

The function write.table() writes an R data frame to a file. Just as read.table() had variants read.csv() and read.csv2(), write.table() has variants write.csv() and write.csv2() to write to a .csv file. To write the **FAT** data frame, stored in the global environment, to the file FAT.txt in R's current working directory, type

```
> write.table(FAT,file="FAT.txt")
```

The previous command, by default, stores the data frame **FAT** to the file FAT.txt using blank spaces as the separators. To store the file using tab separation, type

```
> write.table(FAT,file="FAT.txt", sep="\t")
```

3 Exploring Data

There are lots of functions in R that allow us to explore data. Here we will look at some of the basic ones. The names of these functions are quite intuitive and don't require much explanation. These are: min(), max(), median(), mean(), var(), sd() and summary(). The summary() function includes the results of the first four of these functions and it's a good way of obtaining the five number summary of data. The five number summary can also be obtained using the function fivnum().

Enter the following data set and store the results in the variable taxis. The data set refers to the number of kms that Brand X tyres last before they are worn out, trialled on eight taxis

Taxi 1	Taxi 2	Taxi 3	Taxi 4	Taxi 5	Taxi 6	Taxi 7	Taxi 8
34400	45500	36700	32000	48400	32800	38100	30100

Apply all the functions listed above to the data set. So, for example, to find the mean of the above data you would type

```
> mean(taxis)
```

3.1 Stem and leaf plots

Enter the following data into R and call it mydata: 54, 59, 35, 41, 46, 25, 47, 60, 54, 46, 49, 46, 41, 34, 22. Now type

```
> stem(mydata)
```

3.2 Histogram

The function to draw a histogram is hist(). Look it up in the help menu and explore the different arguments.

We will use the Cars93 data in the "MASS" library which represents the sales of different models of car in the year 1993. We will need to make the MASS library available first, so type

```
> library("MASS")
```

Now draw a histogram of the minimum prices of cars in that data set:

```
> hist(Cars93$Min.Price)
```

Make your plot more interesting by adding colour and changing the x-axis label and the title.

3.2 Box Plots

To create a boxplot, use the command boxplot(). By default, boxplots have a vertical orientation. To create a horizontal boxplot, use the optional argument horizontal = TRUE. Common arguments for boxplot() include col= to set the box color and notch = TRUE to add a notch to the box to highlight the median.

Use the data frame Cars93 in the MASS package to create a boxplot of the variable Min.Price.

In its simplest form we would type:

```
> boxplot(Cars93$Min.Price)
```

Explore other arguments to make the plot more interesting.

Using boxplots to compare variables is easy also. See if you can figure out what the following command does

```
> boxplot(Cars93$MPG.city~Cars93$Type)
```

Exercises

1. Read the **Concrete_Data.csv** file into *R*.
2. Use the **dim()** function to work out the dimensions of the concrete data, i.e. the number of rows and columns.
3. Use the **head()** function to take a look at the first few rows of the concrete data set. The function **tail()** can be used to see the last few rows of a data set. Try it.
4. Produce a five-number summary of the compressive strength of concrete.

5. Plot a histogram of the compressive strengths of concrete. Use an informative title and appropriate *x* and *y* labels. You can also add colours by using the argument *col = "Colour_name"*. (The units of compressive strength are MPa).
6. Produce a boxplot of compressive strengths of concrete. Add an appropriate title and y-label.