LAB100

Week 15: Importing and Summarising Data.

Your aim for this workshop is to learn load data into RStudio, perform basic data manipulation and obtain simple summary statistics from the data. Upon completing this workshop you should be able to:

- Import data in text and spreadsheet formats.
- Manipulate a data set, including deal with missing values.
- Export data to a .csv file.
- Obtain summary statistics such as means, quantiles and medians.

For this workshop you will need to download the following data files from Moodle: ***airquality.txt*** and ***FTSE100.csv***. Remember that you need to change the working directory and open a new R script at the start of each workshop.

1 Importing Data From a *.txt* File

Typing data into RStudio can be a laborious task and is particularly prone to errors when there is a lot of information to import. Instead, if there exists a digital file that contains the information we need, then we can import the data directly. Commonly, data is either stored as a table in a text editor (*.txt*) or as a spreadsheet (*.csv*). Other document formats are possible, but most can be coerced into one of these two types.

We import a *.txt* file into RStudio using the `read.table` command. It is used as follows, where the the commands arguments are described in the table below.

```
read.table( file, header, row.names, col.names, nrows, skip )
```

IMPORTANT: It is essential that you change your working directory to wherever your data is stored. This can be done by going to the "Session" menu tab in RStudio and selecting 'Set Working directory > Choose Directory...'

The information in the file ***airquality.txt*** contains data on air quality measurements over a 5 month period in New York, USA. The measurements include ozone levels (parts per billion), solar radiation (Langleys), wind speed (miles/hour) and temperature (°F). First open the file and you will notice that the rows and column of the table has headings. Therefore, importing this data into RStudio can be executed with:

```
read.table("airquality.txt", header=TRUE)
```

You will now see the contents of the file printed in the console panel.

Argument	Description
`file`	The file name with the file type. This must be enclosed with quotation marks: For example `"file.txt"`
`header`	A logical argument specifying whether the file has row and column headings. Default: `TRUE`.
`row.names`/`col.names`	A vector of text specifying alternative headings for the row and columns respectively.
`nrows`	An integer specifying how many lines of the file should be imported. (Default: the whole file is imported.)
`skip`	An integer specifying how many lines should be ignored before reading to import. This is useful when the file begins with a description of the data. (Default: no lines are skipped.)

Type and run the following commands which illustrate how the command's arguments influence what is imported into RStudio:

```
read.table("airquality.txt", header=FALSE, skip=1)
read.table("airquality.txt", skip=62, nrow=31)
read.table("airquality.txt", skip=62, nrow=31, col.names=c("ID", "Ozone",
    "Solar.Rad", "Wind.Seed", "Temp", "Month", "Day"))
```

The first example skips the first line of the file and loads the remaining contents of the file without headings; notice that the column names have changed. The second example imports the data relating to July (month 7) by skipping the first 62 lines and reading the next 31 lines. The final example imports the same data as previous, but labels the column headings as stated.

Although the `read.table` command is importing the file, it automatically prints the information directly to the console and it is then forgotten. If we wish to use the data, then the imported data must be allocated an object name:

```
airquality <- read.table("airquality.txt")
airquality
```

Computation with the imported data can now be treated similarly as information stored within a matrix array. Therefore, all of the information on the temperature can be extracted by typing `airquality[,4]` into the console. However, with large files with information on many variables it can be difficult to remember which column number relates to which variable. The `names` command helps with managing large data sets by printing the headings of the data frame to the console screen.

```
names( airquality )
```

We can also obtain a particular column of a data frame by referring to it by name using `$`, for instance `airquality$Ozone`.

Another useful command for managing these data frames is `attach`. This enables you to request the contents of a specific variable by typing the column's heading. For example, we can attach the heading of the air quality data set and retrieve the ozone measurements by typing and executing the following:

```
attach( airquality )
Ozone
```

2 Importing Data From a *.csv* File

Alternative to text files, data can be stored within spreadsheets as a *.csv* file. This can be opened by many programs including Microsoft Excel and iWork Numbers. To import this type of file into RStudio we use the `read.csv` command that takes as arguments the `file` name, with the *.csv* suffix and enclosed with quotation marks, and a logical constant stating whether the file has row and column `headers`:

```
read.csv( file, header )
```

The data in the file ***FTSE100.csv*** was downloaded from http://uk.finance.yahoo.com and contains the daily trading summaries for the FTSE100 share index from 1st January 2007 to 31st December 2010. To import this file into RStudio, type the following into your editor and execute:

```
ftse <- read.csv("FTSE100.csv", header=TRUE)
```

Similar to before, we assign the data an object name in order to use the data. Now type `ftse` to print the data to the console. Scroll up the console panel to view the column heading, but you should notice that the top few records have been omitted because there is more data than there is printable space allowed for the console. Instead, we can view parts of the data by running the following:

```
names(ftse)
attach(ftse)
Date
Close
```

3 Exporting Data To a *.csv* File

A useful piece of information for financial analysts is the daily return of a stock and is calculated by the formula:

$$\text{Return}_i = \frac{\text{Close Price}_i - \text{Open Price}_i}{\text{Open Price}_i}$$

In RStudio we calculate this by executing:

```
Return <- (Close - Open)/Open
Export <- cbind( Date, Return )
```

The last line creates a new data frame that we are going to Export to a new spreadsheet containing the calculated daily returns and the corresponding trading date.

To export data to a *.csv* file we use the `write.csv` command, which takes the same arguments as the `write.table` command. Therefore, to export the return data, type the following into your editor and execute.

```
write.csv(Export,"Return.csv")
```

Note that a new file has been created in your working directory called *Return.csv*. Again, be careful when exporting data as RStudio will overwrite any existing file with the specified file name and it will give no warning that it is doing so.

Note that we could also append the `Return` variable to our existing `ftse` dataset.

```
ftse$Return <- Return
```

4 Summary Statistics

Summarising data is important for communicating the key features about a random phenomenon without reporting all of the collected data. For example, a company that sells air conditioning units is unlikely to want to know the temperature on every summer's day, but rather like to know the average or range of temperatures in a particular area to assess whether there will be enough demand for their product to make a profit. RStudio can evaluate many useful summary statistics, some of which are described in the table below:

Command	**Definition**
`mean(x)`	Average value of the vector x
`median(x)`	Median value of the vector x
`quantile(x,p)`	The p% percentile of the vector x
`variance(x)`	Variance of the vector x
`sd(x)`	Standard deviation of the vector x
`table(x)`	Constructs a frequency table
`summary(x)`	Gives the min, max, mean and quartiles of x

First consider the mean, variance and standard deviation commands. The table below shows you how to use these commands to summarise the temperatures from the air quality data set. Compare the the shorthand commands to that of the longhand commands, which are derived from their formulaic definitions.

Code (Shorthand)	**Code (Longhand)**	**Formula**
	`n <- length( Temp )`	
`mean( Temp )`	`MEAN <- sum( Temp )/n`	$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$
	`MEAN`	
`var( Temp )`	`VARIANCE <- sum( (Temp - MEAN)^2 )/( n-1 )`	$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$
	`VARIANCE`	
`sd( Temp )`	`STDDEV <- sqrt( VARIANCE )`	$S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$
	`STDDEV`	

Other summary values are the median, the lower and upper quartiles and the interquartile range. The first of these can be evaluated by the `median` command:

```
median( Temp )
```

To calculate the lower or upper quartiles we use the `quantile` command which accepts two arguments: a numeric vector containing the data and a probability value (a number between 0 and 1). Recall that the lower and upper quartile are the 25% and 75% percentile respectively.

```
l.quartile <- quantile( Temp, 0.25 )
u.quartile <- quantile( Temp, 0.75 )
```

Note that a quartile at 50% returns the same value as the median. The interquartile range is therefore the difference between these two values:

```
u.quartile - l.quartile
```

These commands are useful for when you are interested in a specific summary value. Alternatively, the `summary` command calculates the minimum and maximum, the lower and upper quartiles, the mean and median values in one executable command:

```
summary( Temp )
```

The Pearson correlation coefficient, given by

$$\rho(\mathbf{x}, \mathbf{y}) = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}}$$

for two vectors of length n, $\mathbf{x}$ and $\mathbf{y}$ can be calculated in R using the function `cor`. For instance

```
cor(Temp, Wind)
```

returns a value of -0.458, suggesting a negative association between the two variables, i.e. hot days tend to be less windy.

Finally, if your data is discrete or categorical, it may be more convenient to summarise the data into a frequency table. Type the following into your editor

```
table(Temp )
```

The function `table` can also be used to calculate cross-tabulations of two or more variables. The following code gives the counts of each value of `Temp` in the 5 months of the study

```
table(Temp ,Month )
```

5 Data manipulation

In practice, we may have the data we wish to analyse in quite the form we would like. For instance, suppose that we are only interested in airquality data for the months June, July and August. The dataset provided includes May and September as well. We can use the same methods of *subsetting* of matrices, to the data frames used here. We can make a new data frame consisting only of observations from June, July and August, i.e. where `Month` is equal to 6, 7 or 8.

```
summerairquality <- airquality[airquality$Month%in%c(6,7,8),]
```

Note that if we are manipulating a dataset, or making several variants of it, it is sensible to `detach` the data from the Workspace, so that there are not multiple conflicting versions of the variables.

```
detach(airquality)
```

A further feature of the `airquality` dataset is the presence of missing values. Missing values are denoted by `NA` in R. The presence of missing data is a common problem in practical data analysis. If we attempt to calculate a summary statistic, e.g. the mean, of a data vector containing missing data then, by default at least, we will run into problems:

```
mean(airquality$Ozone)
```

this returns `NA` because several of the entries of `Ozone` are missing. We can obtain the mean of the non-missing elements of the vector by adding the option `na.rm = TRUE` to our command

```
mean(airquality$Ozone,na.rm=TRUE)
```

Note for the function `cor`, the equivalent option is to set `use="complete"`. To check whether a particular observation is missing we can use the function `is.na`. If applied to a vector this will return a logical vector with `TRUE` whenever the observation is missing and `FALSE` otherwise. We can therefore obtain an overall airquality dataset that removes the observations for which `Ozone` is missing by subsetting in the following way:

```
airquality2 <- airquality[!is.na(airquality$Ozone),]
```

More generally, we may want to remove any observations in which any observations are missing, so that a *complete-case* analysis may be performed. We can do this using the function `na.omit`.

```
completeairquality <- na.omit(airquality)
```

The resulting dataset contains 111 rather than 153 observations because the observations where `Ozone` or `Solar.R` were missing have been removed.

Finally, we may need to sort the data in some way. Currently the data is sorted by `Temp`. However, it may be more useful to have the data in chronological order. To do this we can use the function `order`.

```
airquality <- airquality[order(airquality$Month,airquality$Day),]
```

Note that we get a different ordering if we use

```
airquality <- airquality[order(airquality$Day,airquality$Month),]
```

as this sorts first by day of the month and then by month, meaning we get all the 1st of the month data together.

Quiz 1: Cross-tabulation

Using the full airquality dataset, determine how many days in June (`Month`= 6) the temperature was 75 or above.

Quiz 2: Conditional mean

Using the full airquality dataset, calculate the mean wind speed (`Wind`) on days where the temperature was 79 or above. Give your answer to 2 decimal places.

Quiz 3: Association I

Exclude observations with missing `Ozone` and hence calculate the Pearson correlation between `Wind` and `Ozone` to 3 decimal places.

Quiz 4: Association II

Using only observations with complete data on all variables, calculate the Pearson correlation between `Ozone` and `Solar.R` to 3 decimal places.

Quiz 5: Change in temperature

Using the full airquality dataset, construct a new variable defined as the change in `Temp` between consecutive days, for instance `Change` is equal to 5 for `Month=5`, `Day=2` because the temperature went from 67 on Day 1 to 72 on Day 2. Calculate the Pearson correlation between `Change` and `Solar.R`, for days in which both quantities are available, to 3 decimal places.
[NB: The `Change` variable should be missing, i.e. equal to `NA`, for the first day in the dataset (`Month=5, Day=1`).]