INFO411: Data Mining and Knowledge Discovery

R Tutorial and Visualisation

General Instructions: Please Read Carefully

1. This lab consists of two parts:
   - The first part is a general R tutorial. If you are already familiar with R, you can skim it or skip it. An alternative resource that covers similar information is an interactive tutorial at `http://tryr.codeschool.com`.
   - The second part, starting on page 18 is a list of questions that you must use the knowledge from the R tutorial and the lecture to answer and tasks you must perform. In particular, you might want to take a look at the posted R code with detailed comments of what each part of the graphic function call does.
2. The datasets are available either as part of the R packages or on Moodle.
3. Work in a group of size two to three (no more than three students are to work together).
4. Submit a PDF document which contains your answers of the questions in the second part. Your document must be properly typeset (do not submit a scanned version of a handwritten document). One document is to be submitted by each group. The header of the document must list the name and student number of all students in the group.
5. This submission will not be assessed.

1 R and RStudio

RStudio is a powerful statistical analysis package. It requires R to operate, so you must install both R and RStudio onto your computer.

1.1 Installing R

R is available for free download from:

`http://cran.csiro.au`

Within the **Download and Install R** box, there are three options to choose from depending on the operating system that your computer runs.

- **Mac (Apple):**
  1. Click on **Download R for (Mac) OS X**
  2. Under the heading **Files** click on **latest version** and download it.
  3. Go to your downloads folder and click on the downloaded R file (R-latest.pkg) to open it.
  4. Follow the onscreen installation process to install R.
- **Windows (PC):**
  1. Click on **Download R for Windows**.
  2. Click on **install R for the first time**
  3. Click on **Download R *3.x.x* for Windows** at the top of the screen to the download the latest version (*3.x.x*).
  4. Click on the downloaded .exe file to start installation. Most users will want to accept the defaults. The effect is to install the R base system, plus recommended packages. Windows users will find that one or more desktop R icons have been created as part of the installation process.

If you already have R installed on your computer, you can make sure you have the latest version available by installing R as outlined above.

1.2 Installing RStudio

RStudio is available for free download from:

`http://www.rstudio.com`

1. Within the **Powerful IDE for R** box, click on **Download now**.
2. On the next page, under the heading **If you run R on your desktop**, click on **Download RStudio Desktop**.
3. On the next page, under the heading **Recommended For Your System**, click on the file and download it. Notice that your operating system (either Mac OS X or Windows) has been detected and the appropriate version of RStudio is available for you.

4. Click on the icon for the downloaded installation file to install it. An RStudio icon will appear. Click on the icon to start RStudio. RStudio should find any installed version of R, and if necessary start R.

1.3 Installing and loading packages

1. First, start R, perhaps by clicking on an R icon. Make sure that you have a live Internet connection.

2. For this lab, we will be using two packages, the `Hmisc` package and the `MASS` package.

3. To install `Hmisc` (if it isn't already) enter the following in the **Console**:
   ```r
   install.packages("Hmisc", dependencies = TRUE)
   ```

   We include the `dependencies=TRUE` argument to ensure that all other packages that are required by `gdata` are also installed.

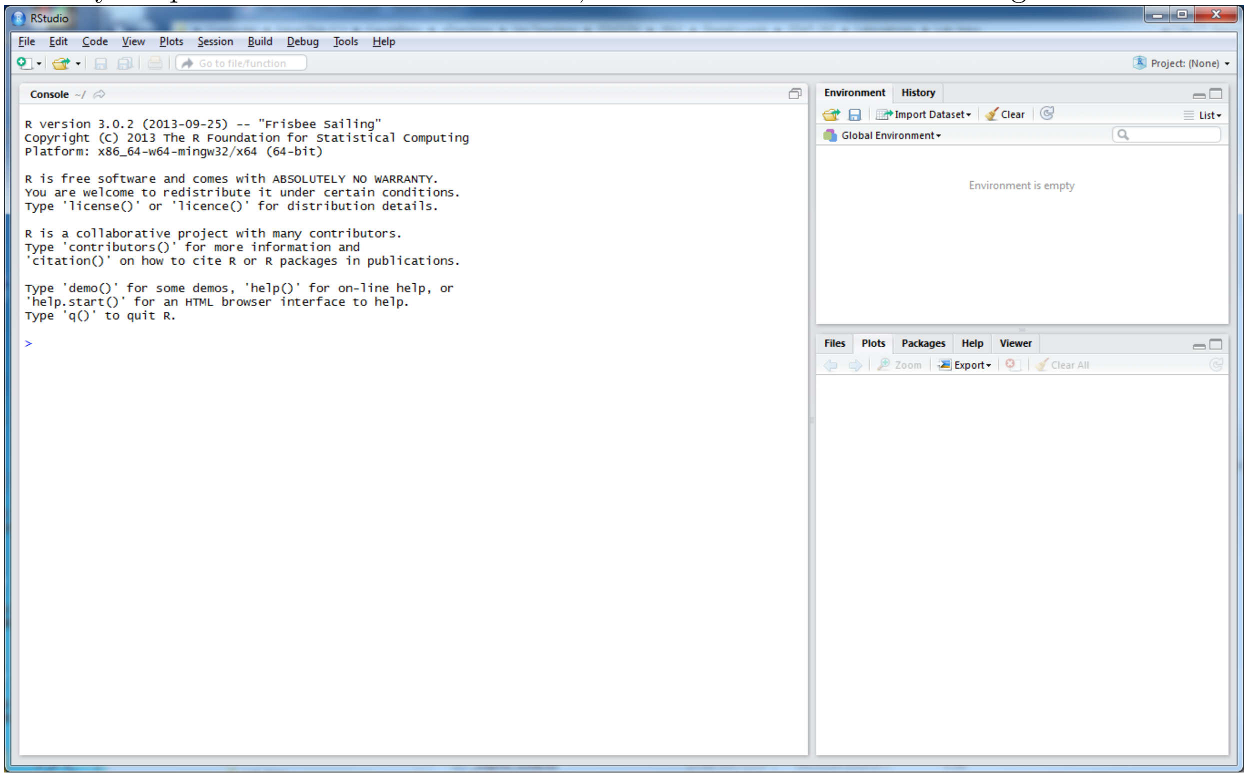
4. To install `MASS` (if it isn't already) enter the following in the **Console**:
   ```r
   install.packages("MASS", dependencies = TRUE)
   ```

5. Every time you open RStudio and wish to use these packages you must first load them into the library. You can do this by:
   ```r
   library(Hmisc)
   library(MASS)
   ```

   Of course if you don't need to use them you don't need to load them. Notice that these packages now have a tick next to them under the **Packages** tab in the bottom right window.

1.4 Opening RStudio

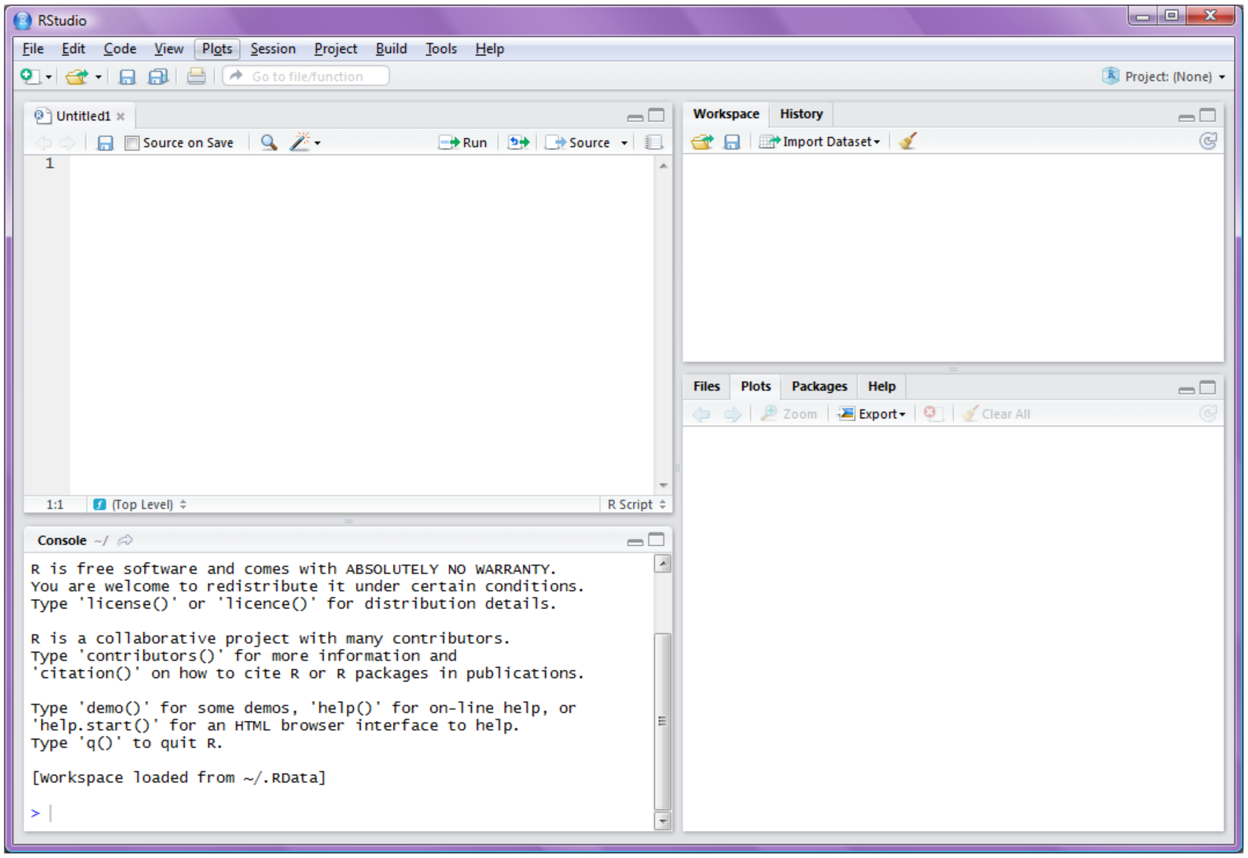
When you open RStudio for the first time, it should look like the following:



```
R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (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.

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.

>
```

1. Click on **File > New > R Script**

The screen now looks like the following:



```
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.

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.

[Workspace loaded from ~/.RData]

> 
```

This will open a new **R Script** in the top left corner. This is where you will type your coding. The advantage of using an **R Script** is that your code will always remain there, like a document, even after it has been run.

In the top right corner is the **Workspace**. This shows the data, matrices, variables, values, etc that have been stored. A **History** of code that has been run is also available here.

In the bottom right corner is where the **Plots** are output and **Help** is given. This window also gives access to the **Files** that are stored on the computer and **Packages** that have been installed in RStudio.

Finally, The **Console** is in the bottom left corner. You can also run code in this window, but it is not as convenient as using the **R Script**. This window also displays the output from the code that has been run in the **R Script**.

1.5 Use of the R Script window

1. Return to the **R Script** and type:

   ```
   2 + 2
   ```

   You can either run the line that the cursor is in or a selection of lines by highlighting them. To run items in the **R Script**, hold $\boxed{\text{CONTROL}}$ and then hit $\boxed{\text{ENTER}}$ at the same time. Alternatively, hit the **Run** icon that is situated to the top right of the **R Script** window. Now look in the **Console** window, as mentioned this is where the output from the code run in the **R Script** is given. The following appears on screen:

```
2 + 2

## [1] 4
```

In the **Console**, the code that has been run and the result from the code is shown. The `[1]` says "first requested element will follow". In this case we have requested just one element, the solution to 2+2. The `>` indicates that R is ready for another command.

2. Next, we create an object containing the solution to 2+2. In the **R Script**, hit $\boxed{\text{ENTER}}$ to start a new line. The number 2 will appear in the left hand margin to indicate the 2nd line of code. Run the following code on the 2nd line of the R Script.

```
Result_1 <- 2 + 2
```

The assignment symbol is `<-` and we use it to store items in objects. The value 4 is now stored in an object with the name `Result_1`. You can check this by either looking in the **Workspace** or by typing `Result_1` into the third line of the **R Script** and running it. R is case sensitive, so remember to use capitals where applicable. The following appears in the **Console** after typing in `Result_1`:

```
Result_1 <- 2 + 2
Result_1  # Check the contents of 'Result_1'

## [1] 4
```

Notice the sentence `# Check the contents of 'Result_1'`, this is a comment and is not run by RStudio. It is the `#` symbol that indicates what follows, on that line, is comment. It is always helpful to include comments in your work for future reference, but it is probably not necessary in this Lab. Objects in RStudio include vectors, in this case we can treat `Result_1` as a vector with only one element. But next we will consider a vector with multiple elements.

3. Vectors with multiple lengths can be created and checked by running:

```
Vector_1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
Vector_1  # Check the contents of 'Vector_1'

##  [1]  1  2  3  4  5  6  7  8  9 10

Vector_2 <- c(11, 12, 13, 14, 15, 16, 17, 18, 19, 20)
Vector_2  # Check the contents of 'Vector_2'

##  [1] 11 12 13 14 15 16 17 18 19 20
```

It is the `c` that is important here. This indicates that a vector follows, and must be included when creating all vectors. Without the `c` RStudio produces an error as it does not recognise what is going on.

Hint: You could use `Vector_1 <- 1:10` because in this case the vector is just a sequence from 1 to 10 that goes up by one. Alternatively, we could create any sequence via:

```
seq(from, to, by)
```

and we can specify any start point (`from`), end point (`to`), and step size (`by`) we like.

1.6 Saving your work

1. Save the R script by clicking on **File > Save**
2. Specify the file name `Intro_Lab` and set the folder to your storage medium of choice, click **Save**.
3. When you open RStudio again, you can open your saved R Script by clicking **File > Open File...** then find the file in the folder.

1.7 Basic data structures and operations

1. Now that you have created vectors, there are many operations that you can perform on them. For instance,

```
# Obtain the second element of a vector
Vector_1[2]

## [1] 2

# Obtain second through fifth elements of a vector
Vector_1[2:5]

## [1] 2 3 4 5

# Obtain the third, the sixth, and the second element of a
# vector in that order
Vector_1[c(3, 6, 2)]

## [1] 3 6 2

# Obtain all but the second element of the vector
Vector_1[-2]

## [1]  1  3  4  5  6  7  8  9 10

# Obtain all but the second through fifth elements of the
# vector
Vector_1[-(2:5)]

## [1]  1  6  7  8  9 10
```

```
# Find which values of 'Vector 1' are greater than or equal
# to 4
Vector_1 >= 4

##  [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE

# (Other relational operators are: >, <=, ==, != indicating
# greater than, less than or eual to, equal to, and not
# equal.)

# Find which values are strictly less than 4
Vector_1 < 4

##  [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

# or, logically negate with !
!(Vector_1 >= 4)

##  [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

# Find their positions
which(Vector_1 >= 4)

## [1]  4  5  6  7  8  9 10

# Obtain those elements of Vector_2 whose corresponding
# elements of Vector_1 >= 4
Vector_2[Vector_1 >= 4]

## [1] 14 15 16 17 18 19 20

summary(Vector_1)  # Produce a summary of 'Vector_1'

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    3.25    5.50    5.50    7.75   10.00

mean(Vector_1)  # Find the mean of the values in 'Vector_1'

## [1] 5.5

sd(Vector_1)  # Find the standard deviation of 'Vector_1'

## [1] 3.02765

sum(Vector_1)  # Find the sum of the values in 'Vector_1'

## [1] 55

length(Vector_1)  # Check the length of 'Vector_1'

## [1] 10

length(Vector_2)  # Check the length of 'Vector_2'
```

```
## [1] 10
```

Basic arithmetic can be done on the vectors as they are the same length. For instance,

```
Vector_1 + Vector_2  # Add the vectors together

##  [1] 12 14 16 18 20 22 24 26 28 30

Vector_1 * Vector_2  # Multiply the vectors together

##  [1]  11  24  39  56  75  96 119 144 171 200
```

Notice that the operations are performed on the corresponding elements within each object and that the output also has 10 values.

2. If we were to perform basic arithmetic between `Vector_1` and `Result_1` we would get:

```
Result_1 + Vector_1  # Add together

##  [1]  5  6  7  8  9 10 11 12 13 14

Result_1 * Vector_1  # Multiply together

##  [1]  4  8 12 16 20 24 28 32 36 40
```

and we can see that the value of `Result_1` is applied to every element in `vector_1` according to the operation. The output again has 10 values.

1.8 Matrices and arrays

1. In R, matrices and arrays are vectors given dimension

```
# matrix() constructs a matrix with a given number of rows
# and columns from a vector
Matrix_1 <- matrix(Vector_1, 2, 5)
Matrix_1

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10

# Notice that the matrix is filled by columns. An optional
# byrow argument can be used to control that:
matrix(Vector_1, 2, 5, byrow = TRUE)

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10

# You can inspect their dimension
dim(Matrix_1)

## [1] 2 5
```

```
# Elements can be accessed using brackets, but note the
# comma:

# Element in the first row of the first column
Matrix_1[1, 1]

## [1] 1

# Whole first row (dimension 1)
Matrix_1[1, ]

## [1] 1 3 5 7 9

# Whole first column (dimension 2)
Matrix_1[, 1]

## [1] 1 2

# Arrays can have three or more dimensions
Array_1 <- array(1:24, c(3, 4, 2))
Array_1

## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,]   13   16   19   22
## [2,]   14   17   20   23
## [3,]   15   18   21   24
```

2. Some functions are designed to work on matrices and arrays

```
# Find the row sums of Matrix_1; note the last argument is a
# function that gets evaluated on each row
apply(Matrix_1, 1, sum)

## [1] 25 30

# Find the column sums of Matrix_1
apply(Matrix_1, 2, sum)

## [1]  3  7 11 15 19

# Find the sum over the 3rd dimesion of the array
```

```
apply(Array_1, c(1, 2), sum)

##      [,1] [,2] [,3] [,4]
## [1,]   14   20   26   32
## [2,]   16   22   28   34
## [3,]   18   24   30   36

# compare:
Array_1[, , 1] + Array_1[, , 2]

##      [,1] [,2] [,3] [,4]
## [1,]   14   20   26   32
## [2,]   16   22   28   34
## [3,]   18   24   30   36
```

1.9 Data frames – grouping together data

1. If you have multiple objects of the same length, you can combine them into a table or what is formally known as a *data frame*. A data frame not only gives you the ability to store your data in a table, but is the preferred way to make data available to modelling functions and plots. Create a data frame out of the two vectors by:

```
Dataframe_1 <- data.frame(Vector_1, Vector_2)
# Check the contents of 'Dataframe_1'
Dataframe_1

##    Vector_1 Vector_2
## 1         1       11
## 2         2       12
## 3         3       13
## 4         4       14
## 5         5       15
## 6         6       16
## 7         7       17
## 8         8       18
## 9         9       19
## 10       10       20

# You can also use head() to quickly inspect its first few
# rows:
head(Dataframe_1)

##   Vector_1 Vector_2
## 1        1       11
## 2        2       12
## 3        3       13
## 4         4        14
## 5         5        15
## 6         6        16
```

You can also check the contents of `Dataframe_1` by clicking on it in the **Workspace**.

2. You also have the ability to change the column names by:

   ```
   colnames(Dataframe_1) <- c("A", "B")
   ```

   Again check the new column names of `Dataframe_1` by clicking on it in the **Workspace**. Notice that as there were two column names we created a vector containing them using `c`. We quoted `"A"` and `"B"` to indicate to R that they are text not objects.

3. Now that we have created a data frame, we have the ability to access different parts of it. For instance, the following all refer directly to the 2nd column of the data frame.

   ```
   # Access the 2nd column by specifying the column name after £
   Dataframe_1$B

   ##  [1] 11 12 13 14 15 16 17 18 19 20

   # Specify all elements of the 2nd column by
   Dataframe_1[, 2]

   ##  [1] 11 12 13 14 15 16 17 18 19 20

   # Specify all elements of the column named B
   Dataframe_1[, "B"]

   ##  [1] 11 12 13 14 15 16 17 18 19 20

   # Treat 'Dataframe_1' as a list and access all elements under
   # B
   Dataframe_1[["B"]]

   ##  [1] 11 12 13 14 15 16 17 18 19 20
   ```

4. You may be interested in accessing a particular element within the data frame. For instance, the following all refer directly to the third element of the 2nd column.

   ```
   # Access the third element of B
   Dataframe_1$B[3]

   ## [1] 13

   # Specify the 3rd element of the 2nd column by
   Dataframe_1[3, 2]

   ## [1] 13

   # Specify the 3rd element of the column named B
   Dataframe_1[3, "B"]

   ## [1] 13
   ```

```
# Treat 'Dataframe_1' as a list and view the 3rd element
# under B
Dataframe_1[["B"]][3]

## [1] 13
```

Try to keep it simple so I suggest using either the first or second methods to call on elements as will be the case in the notes.

5. To save typing when working with a table, you can "attach" it so that its column names become variables until it's detached.

```
attach(Dataframe_1)
A  # Returns column A

##  [1]  1  2  3  4  5  6  7  8  9 10

detach(Dataframe_1)
A  # Error!

## Error in eval(expr, envir, enclos):  object 'A' not found

# with() function with can attach a table to execute a just
# one command...
with(Dataframe_1, A + B)

##  [1] 12 14 16 18 20 22 24 26 28 30

# or even several, grouped together with braces ({}) (and
# separated by semicolons (;) if they are on the same line)
with(Dataframe_1, {
    tmp <- A + B
    sum(tmp * tmp)
})

## [1] 4740
```

6. Rows (subsets) of a data frame can also be accessed. The following all obtain the rows for which column `A` is `>= 4`.

```
# Get the logical vector for whether that element of A >= 4,
# and access those rows
Dataframe_1[Dataframe_1$A >= 4, ]

##     A  B
## 4   4 14
## 5   5 15
## 6   6 16
## 7   7 17
## 8   8 18
## 9   9 19
## 10 10 20

# subset() function lets save typing by treating columns as
# variables
subset(Dataframe_1, A >= 4)

##     A  B
## 4   4 14
## 5   5 15
## 6   6 16
## 7   7 17
## 8   8 18
## 9   9 19
## 10 10 20
```

1.10 Basic flow control: loops and conditionals

1. Loops are slow in R so it is best to avoid them when possible. However, sometimes it is necessary to loop through iterations in sequence. The basic syntax is shown in the following example. Note that simply entering the name of an object does not display it when executed inside a loop, so the `print()` function is needed for that purpose.

```
# Iterate through elements of Vector_2
for (i in Vector_2) {
    print(i)
}

## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
## [1] 19
## [1] 20
```

2. The R syntax for `if` statements is as follows:

```
if (2 + 2 == 4) print("correct") else print("wrong")

## [1] "correct"

if (2 + 2 == 5) print("correct") else print("wrong")

## [1] "wrong"
```

1.11 RStudio Help

RStudio has an extensive help system, every command and function has it's own help page. Search through the help pages via the **Search Engine & Keywords** icon under the **Help** tab in the bottom right window. If you know the name of the function that you need help with, insert a `?` followed by its exact name in the **Console** window and hit $\boxed{\text{ENTER}}$. Type

```
? mean
```

in the **Console** to obtain the RStudio help file on the `mean` function.

1.12 Input of data from a file

The function `read.table` is used to input data from a file into a data frame. As an example we will consider the data file `plane_excel.csv` found on Moodle. Save this file along with the remaining data files for this lab into a working folder on the storage medium of your choice.

1. Return to RStudio and set the working directory to the folder to which you saved the files. To do this, click on

   Session > Set Working Directory > Choose Directory...

   In the browser find the folder, and choose it as the working directory. In the **Console** will appear `setwd()` and the parentheses will contain the directory to the folder.

2. Now, the **Files** tab in the bottom right hand window will contain all files that are the directory to which you save them. You should view the data file prior to reading it into R.

3. Click on `plane_excel.csv`, and a new tab will open next to the *R Script*. Notice that the headings of each column are situated in the first row, and how the two columns are separated by a comma. This is because this type of file contains comma-separated values (hence `csv`).

4. Return to the R Script and use the `read.table` function to read in the plane data and assign it to `Data_1`.

   ```
   Data_1 <- read.table("plane_excel.csv", sep = ",", header = TRUE)
   ```

   The first argument within the `read.table` command specifies the file name that contains the data (make sure to type it exactly as shown). Next we use `sep=","` to let R know that the elements are separated by a comma. Finally, set `header=TRUE` to let R know that the first row contains the column headings. Check the data frame `Data_1` by clicking on it in the **Workspace**. Upon observation it has 45 rows and 2 columns.

   You can confirm this by:

   ```
   dim(Data_1)

   ## [1] 45  2
   ```

5. Now we will read in the same data file using the `read.csv` function. This time assign it to `Data_2`.

   ```
   Data_2 <- read.csv("plane_excel.csv")
   ```

   The first argument within the `read.csv` command again specifies the file name that contains the data. But there is no need to use `sep=","` or `header=TRUE` as these are defaults set for this function anyway. Again check the data as outlined in Step 4. Essentially we have created two data frames, `Data_1` and `Data_2`, that are identical.

6. Now view the sulphur oxide data by clicking on `sulphur_oxide.csv` within the **Files** window. A new tab will once again appear next to the R Script. Notice that there are four columns, but we are only interested in the first. The headings are again situated on the first row, but this time the four columns are separated by spaces not commas. The `NA`'s indicate that no values are present. We will again use the `read.csv` function, but will specify the separator as `sep=" "`. Return to the R Script and type:

   ```
   Sulphur_Data <- read.csv("sulphur_oxide.txt", sep = " ")
   ```

7. To remove any unwanted columns, use a minus in front of the column number.

   ```
   Sulphur_Data <- Sulphur_Data[, -2]
   ```

   This has now removed the `emissions.sorted` column. Repeat this procedure twice to also remove both the `Bins` and `Bins2` columns. Notice that the `Sulphur_Data` is now a vector as it only contains the emissions column. Also note that the heading `Emissions` has also gone.

1.13 Saving your work

R Code Save the R script by clicking on **File** > **Save** or the picture of the floppy disk

Workspace environment (the R objects)

To save:

1. Save the **Environment** so that you don't have to run your code when opening RStudio again.
2. Click on the floppy disk icon under the **Environment** tab.
3. Specify the file name and folder, click **Save**.
4. After you have saved the workspace, clear it by clicking the broom icon to the right of the **Environment** tab.

To restore:

1. Open the saved Workspace by clicking on the open icon to the left of the **Environment** tab (brown folder with a green arrow coming out of it).

2. Find the file you had saved it to.

3. The workspace is populated with the objects from this Lab. Do the same process when reopening RStudio so you don't have to run your code.

Console (R output) You can save the **Console** by copying and pasting it into a word document (or similar).

Alternatively, you can save the **Console** by typing the following code into the **Console**.

1.
```r
# Create a log of the Intro Lab
con <- file("Intro_Lab.log")
sink(con, append = TRUE)
sink(con, append = TRUE, type = "message")
# Specify your saved R Script
source("Intro_Lab.R", echo = TRUE, max.deparse.length = 10000)
sink()
sink(type = "message")
```

2. You can get to a new line in the console without running the code by holding SHIFT and then hitting ENTER at the same time.

3. Run this code.

4. A text file containing a log of the code and output from your saved R Script has now been created.

5. You can view it in the **Files tab** in the bottom right window under `Intro_Lab.log` after you refresh the tab via clicking the circular arrow to the far right.

Graphics

To Clipboard (e.g., for pasting into Word):

1. RStudio stores your recent plots. In the **Plots** tab, use left and right arrows in the upper-left-hand corner to select the plot you want to copy.

2. Click **Export▼**, and select **Copy to Clipboard...**.

3. A window labeled **Copy Plot to Clipboard** will appear. Click and drag the triangle in the lower-right-hand corner of the window ◢ to set the desired size and aspect ratio of the image to be copied.

4. Click **Copy Plot**.

5. Paste it into the target application as usual.

1.14 References

Maindonald, J., H. (July 2013) *Data Analysis, Graphics, and Visualisation Using R*.

Exercises

Question 1 Write a loop to spell out the letters of your family name one letter at a time.

Question 2 A dataset available in the `rattle.data` package contains a year of daily obser-vatons of weather from a station in Canberra, with the column descriptions available in the help file. We wish to predict the target variable `RainTomorrow` (i.e., whether it rained the next day). Using the code from the lecture and above:

1. Load the dataset into R by installing `rattle.data`, loading the package, and then typing `data(weather)`. Look at its help file by typing `help(weather)`.
2. Make parallel (side-by-side) boxplots of `MinTemp` grouped by `RainTomorrow` using the function `boxplot`. Does the lowest temperature attained that day appear to be associated with whether it rains the next day?
3. Using the `hist` function, make a histogram of the variable `Sunshine`. Comment on the distribution. (I.e., unimodiality vs. bimodality, skewness, outliers, etc.)
4. Using the `table` function, make a contingency table from the `RainTomorrow` and `WindGustDir` variables in the `weather` data. Then, using this table, make a mosaic plot of `WindGustDir` against `RainTomorrow`. Which orientation (i.e., which variable on the horizontal and which on vertical) is more informative in this case? (Hint: Transpose the table to change the orientation). Comment on any patterns you observe.
5. Using the `pairs` function, make a scatterplot matrix of all the quantitative variables in the dataset, colouring the points according to whether it rains the next day. Comment on any major patterns you see. In particular,

   (a) Which pairs of variables appear to have the strongest correlation? Why do you think they do?

   (b) Which variables have the best "separation" between rainy and non-rainy tomorrows? That is, which variables are good discriminators for rain on the following day irrespective of the other variables?

   Hint: If the dataset has been loaded into a data frame named `weather`, you can extract just the quantitative variables using the following code:

```
weather[, sapply(weather, is.numeric)]
```