University of Wollongong
School of Mathematics and Applied Statistics

STAT231 Probability and Random Variables 2014

Introductory Laboratory

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. In the Horner lab at Uni it is already installed.

1.1 Installing R

R is available for free download from CSIRO domain:

http://cran.csiro.au

Also it is available on CRAN:

http://cran.r-project.org

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 3.0.2 for Windows** at the top of the screen.
    4. Click on the downloaded file to start installation (R-3.0.2-win.exe). 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 Console, perhaps by clicking on an R icon (R x64 3.0.2), or click on **Windows Start Up Menu > R > R x64 3.0.2.**
   Make sure that you have a live Internet connection at home.
2. There are several packages to install for STAT231, for example the *foreign* package and the *plotrix* package.
3. To install *foreign* enter the following in the **Console**:

```
> install.packages("foreign", dependencies=TRUE)
```

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

4. This works on your home computer with live internet connection, first you have to choose a mirror, standard choice is ”Australia”, e.g. ”Canberra”.

```
install.packages(”foreign”, dependencies=TRUE)
Warning in install.packages :
cannot open: HTTP status was ’407 Proxy Authentication Required’
Warning in install.packages :
unable to access index for repository http://cran.rstudio.com/bin/windows/contrib/3.0
```

5. However at uni, R requires proxy-authentication. First, open the homepage: https://www-static.uow.edu.au/cgi-bin/firewallopen.cgi
   and enter your username and password to authenticate yourself.

6. Note this command will not work without successful border authentication. Not all students might have the privilige having the option of being authenticated.

   Now you can return to R and enter—

```
install.packages(”foreign”, dependencies=TRUE)
Installing package into ’C:/Users/tsuesse/Documents/R/win-library/3.0’
(as ’lib’ is unspecified)
— Please select a CRAN mirror for use in this session —
trying URL ’http://cran.csiro.au/bin/windows/contrib/3.0/foreign0.8 − 59.zip′
Contenttype′application/zip′length287435bytes(280Kb)
openedURL
downloaded280Kb
package′foreign′successfullyunpackedandMD5sumschecked
ThedownloadedbinarypackagesareinC : 1nb0packages
```

7. Alternatively (to border authentication) you can download the zip-file of the package, e.g. ”foreign.zip” (use google to obtain, e.g. keyphrase ”R package foreign” or download via www.r-project.org/). Then click on **R** > **Packages** > **Install files from local zip files ...** to select your zip-file. The package will not be automatically installed.

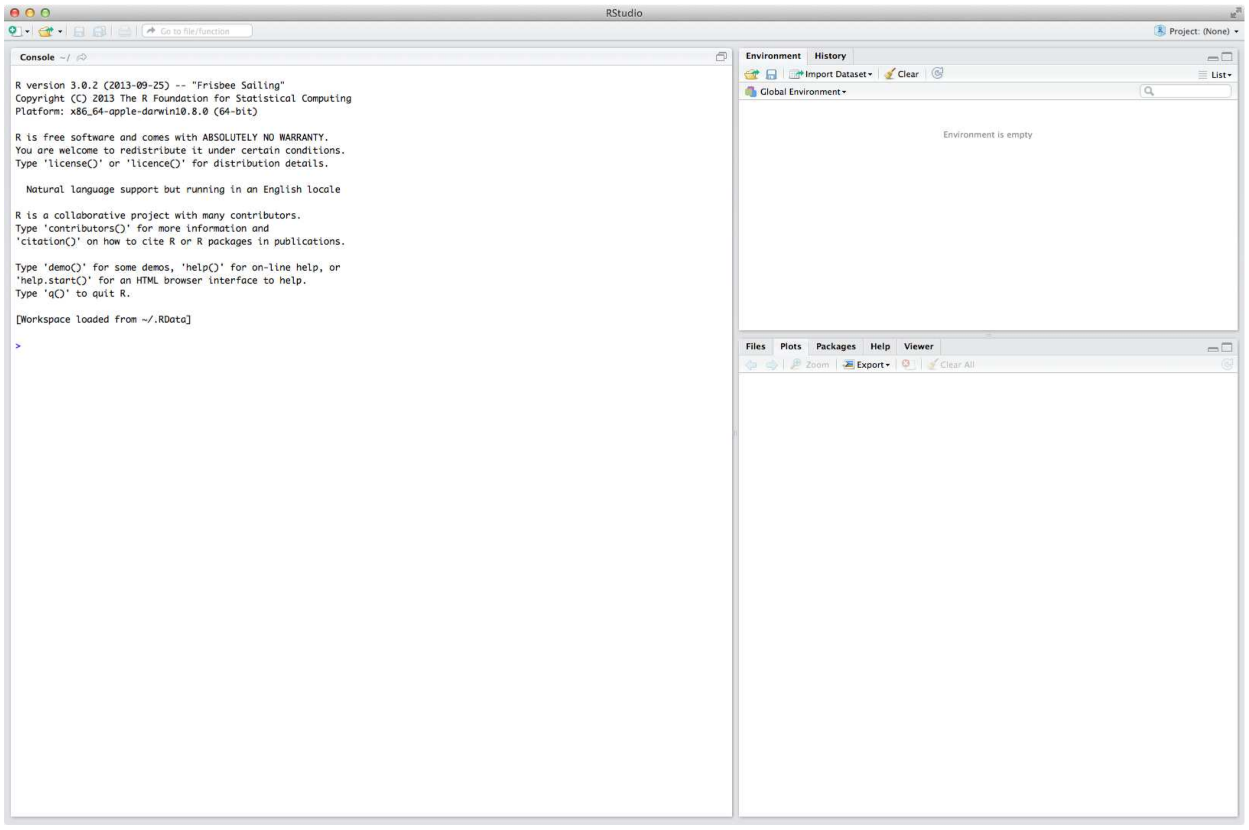
8. To install *plotrix* enter the following in the **Console**:

```
> install.packages("plotrix", dependencies=TRUE)
```

9. Every time you open RStudio (or R Console) and wish to use these packages you must first load them into the library. You can do this by:

```
> library(foreign)
> library(plotrix)
```

   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.



```
R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin10.8.0 (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.

  Natural language support but running in an English locale

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]

>
```

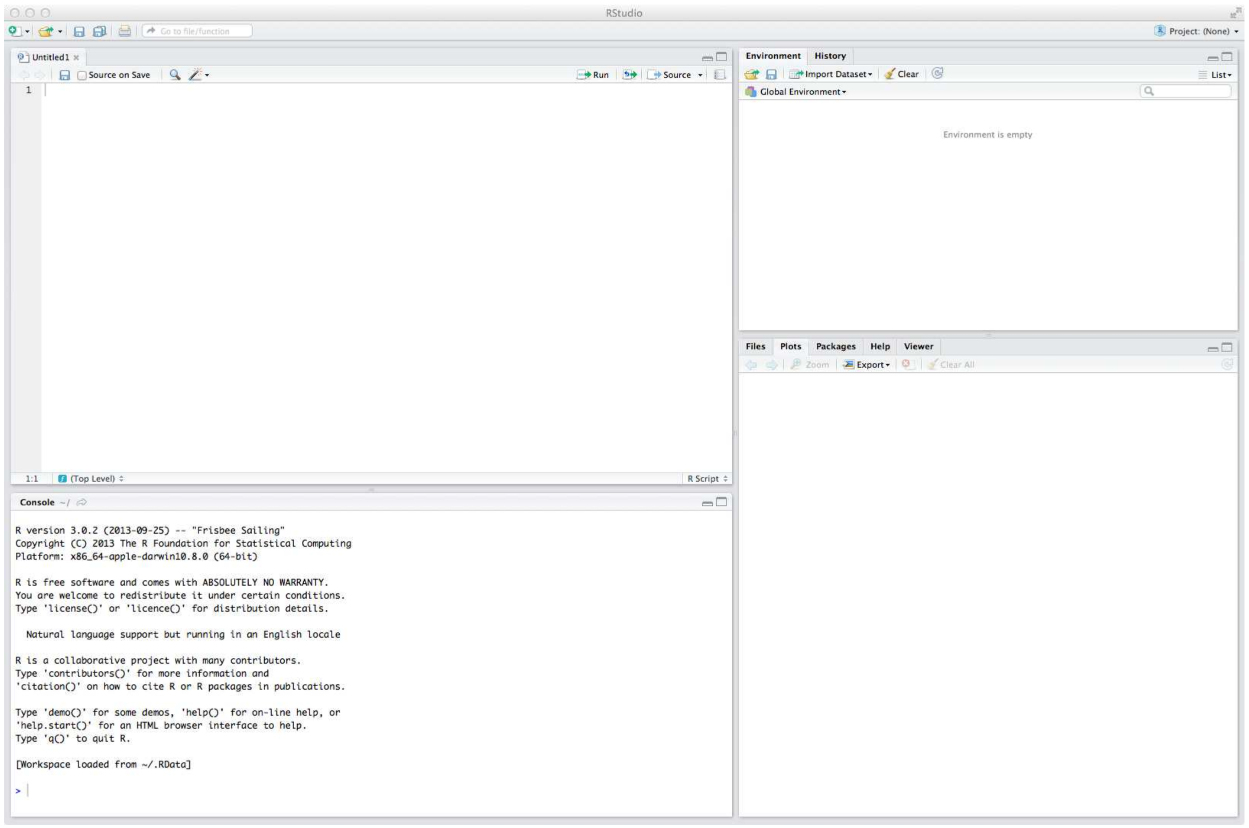
Figure 1: *RStudio opened for the first time in Windows/Mac.*

1.4 Opening RStudio

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

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

The screen now looks 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-apple-darwin10.8.0 (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.

  Natural language support but running in an English locale

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]

>
```

Figure 2: *RStudio with a R Script open.*

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 Installing packages in RStudio

RStudio provides an interactive environment for installing packages, which works fine from home. However setting up ”proxy authentication” at uni is complicated. Note: RStudio uses R (Console) and installing packages under R automatically makes these packages available under RStudio. We will use mainly RStudio at uni and most of the packages are pre-installed, however if some are not installed, please install R packages as described under 1.3 in the R Console and then start RStudio.

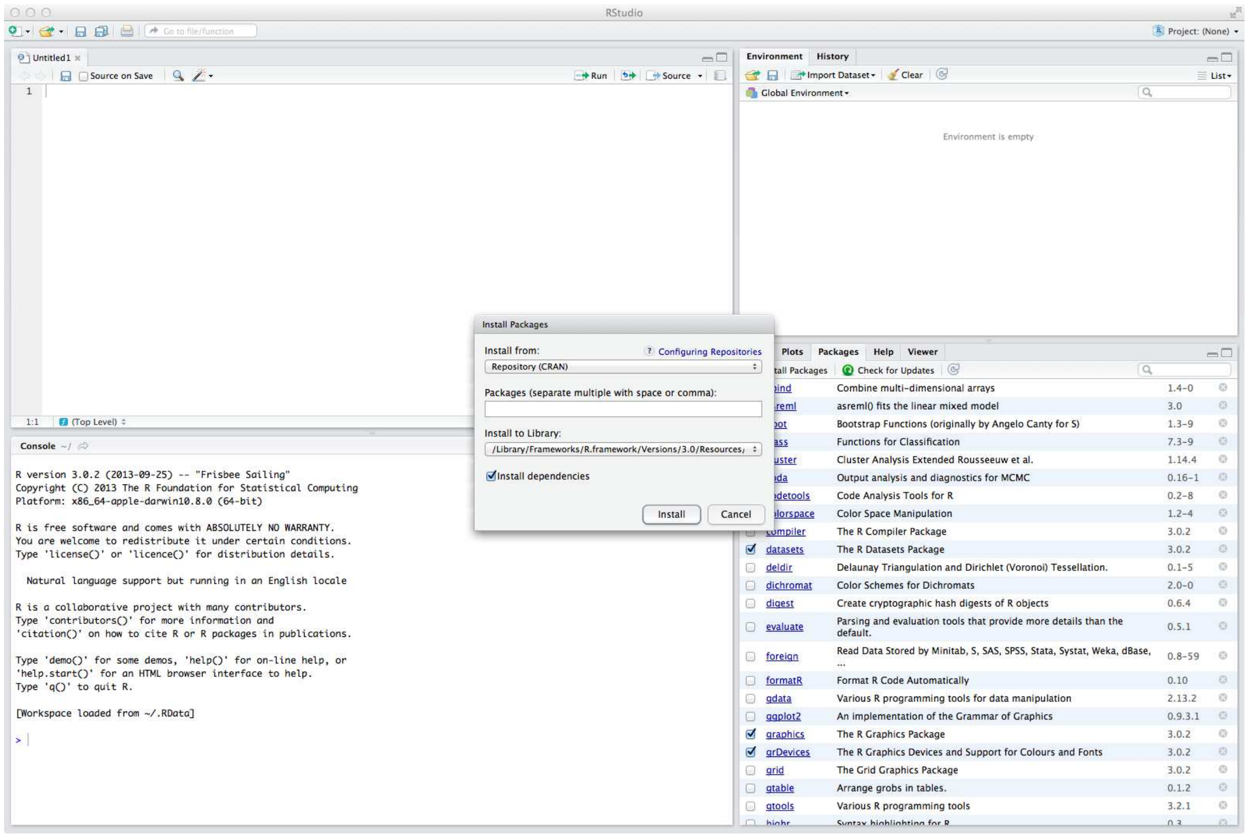
Installing packages in Rstudio (from home) can be done from home as follows:

1. In the package panel which located at bottom left by default, click on

   Packages > Install Packages

2. Using the *Install from* pop-up menu select the an appropriate Repository (CRAN)

3. Type the packages name in the Packages textbox then click on **Install**



Install Packages

Install from: Repository (CRAN)

Packages (separate multiple with space or comma):

Install to Library: /Library/Frameworks/R.framework/Versions/3.0/Resources/

[✓] Install dependencies

```
R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin10.8.0 (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.

  Natural language support but running in an English locale

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]

> 
```

	ind	Combine multi-dimensional arrays	1.4-0
	reml	asreml() fits the linear mixed model	3.0
	oot	Bootstrap Functions (originally by Angelo Canty for S)	1.3-9
	ass	Functions for Classification	7.3-9
	uster	Cluster Analysis Extended Rousseeuw et al.	1.14.4
	da	Output analysis and diagnostics for MCMC	0.16-1
	detools	Code Analysis Tools for R	0.2-8
	lorspace	Color Space Manipulation	1.2-4
	compiler	The R Compiler Package	3.0.2
✓	datasets	The R Datasets Package	3.0.2
	deldir	Delaunay Triangulation and Dirichlet (Voronoi) Tessellation.	0.1-5
	dichromat	Color Schemes for Dichromats	2.0-0
	digest	Create cryptographic hash digests of R objects	0.6.4
	evaluate	Parsing and evaluation tools that provide more details than the default.	0.5.1
	foreign	Read Data Stored by Minitab, S, SAS, SPSS, Stata, Systat, Weka, dBase, ...	0.8-59
	formatR	Format R Code Automatically	0.10
	gdata	Various R programming tools for data manipulation	2.13.2
	ggplot2	An implementation of the Grammar of Graphics	0.9.3.1
✓	graphics	The R Graphics Package	3.0.2
✓	grDevices	The R Graphics Devices and Support for Colours and Fonts	3.0.2
	grid	The Grid Graphics Package	3.0.2
	gtable	Arrange grobs in tables.	0.1.2
	gtools	Various R programming tools	3.2.1
	highr	Syntax highlighting for R	0.3

Figure 3: *RStudio with interactive packages installation menu*

1.6 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. Everything after # symbol is a comment and is not executed by R. It is always helpful to include comments in your work for future reference and self-explanation, 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.

1.7 Saving your work

1. (i) Save the R script by clicking on **File** > **Save**

   (ii) Specify the file name *Intro_Lab.R* (make sure suffix *.R or *.r is used, where * stands for name of file, otherwise RTsudio does not recognise your script as R script and syntax colouring is not available) and set the folder to *Stat231_Intro_Lab*, click **Save**.

   (iii) When you open RStudio again, you can open your saved R Script by clicking **File** > **Open File...** then find the file in the folder *Stat231_Intro_Lab*.

1.8 Basic operations

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

```
> # 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.
> 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 (sample) mean of the values in 'Vector_1'

[1] 5.5

> sd(Vector_1)   # Find the (sample) 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.

1.9 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)
> Dataframe_1             # Check the contents of '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 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. It is also interest to access a part of the data frame. For instance, the interested in a part of vector "B" in which the correspond values of vector "A" are greater and equal 5.

5. 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 a part of vector "B" in which the correspond values of
> # vector "A" are greater and equal 5
> Dataframe_1[,"B"][Dataframe_1[,"A"] > 5]

[1] 16 17 18 19 20
```

```
> # 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.

6. Moreover, it is possible to convert a dataframe to a matrix and vice versa. The following commands refer directly to this conversion.

```
> # Conversion dataframe to a matrix
> Matrix.Data <- as.matrix(Dataframe_1)
> Matrix.Data

      A  B
 [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

> # Conversion matrix to a dataframe
> Dataframe_1 <- as.data.frame(Matrix.Data)
```

1.10 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 it's 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.

1.11 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 moodle. Save this file along with the remaining data files for this lab into a folder named *Stat231_Intro_Lab*.

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

   Session > Set Working Directory > Choose Directory...

   In the browser find the folder with the introductory lab data files, *Stat231_Intro_Lab*, and choose it as the working directory. In the **Console** will appear *setwd()* and the brackets will contain the directory to the folder.

2. now the **Files** tab in the bottom right hand window will contain all files that are in *Stat231_Intro_Lab*. You should view the data file prior to reading it into R.

3. Click on *plane_excel.csv*, 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.tex* 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.

The main point to remember is to first observe the data. Depending on what type of data you have will determine how it should be read into RStudio. We were able to use the *read.csv* and *read.table* functions for the three different data sets as we first observed the data and determined the format of each. Also, there are a number of database packages available which are quite helpful to manage the big dataset. One of these packages is *RODBC* which is very useful for big dataset in excel file format with multi-sheets.

1.12 Basic/Exploratory data analysis

The specific data analysis that is required will depend on the researcher and aims of the research being conducted. Having said this, there are a number of basic analytic techniques that you should become familiar with despite the research being conducted. One of these is:

1.12.1 Always plot the data

There are many different types of plots available in RStudio, depending on the type of data what you wish to observe will determine which plot is appropriate.

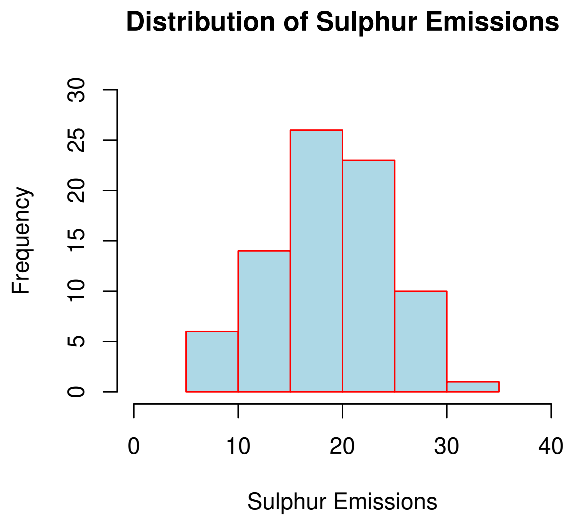
- Recall the sulphur oxide emissions data contained in the object *Sulphur_Data*. This is a single vector containing emissions values.

1. Possibly the best way to plot this is a histogram using the *hist* function.

```
> hist(Sulphur_Data)
```

2. There are a number of aspects that we can specify when plotting graphs. Some aspects are common to all types of plots, others are unique. The title can be set using the argument *main*. Similarly, the x and y labels can be specified by *xlab* and *ylab* respectively. The x and y axis limits can be set by *xlim* and *ylim* respectively. These are in the format of a vector, with the first element indicating the lower limit followed by the upper limit in the last element. Finally, the number of cells (or bars) in the histogram can be set by the *breaks* argument.

```
> hist(Sulphur_Data, main= "Distribution of Sulphur Emissions",
+       col="lightblue", border="red", density= NULL,
+       xlab= "Sulphur Emissions", ylab= "Frequency",
+       xlim= c(0,40), ylim= c(0,30), breaks= 5)
```



Now adjust the breaks from 5 to 10 to 20 and finally to 40 (also change the upper limit of *ylim* from 30 to 15 to 8 to 6 respectively). Take note of what is going on. Notice how the shape of the histogram changes shape. You have to be careful when interpreting histograms as they are very easily influenced by the bar width. Commonly they are not presented alone in a report, but rather as a part of other analysis and statistical tests.

To save a histogram

(i) Click on the **Export** button situated at the top of the **Plots** window. You have the option of saving as either an image or pdf.

(ii) Click on **Save Plot as Image**

(iii) Change the **Image format** to *PNG* and specify the **File name**, click **Save**. (The saved plot will now appear in the **Files** tab in the bottom right window after you click the **Refresh file listing** icon- its a circular arrow on the far right).

(iv) Using RStudio it is possible to save plot in Scalable Vector Graphics (SVG) which provides a super-high quality plot. To do this in RStudio, click on **Plots** > **Export** > **Save Plot as Image** then in format pop-up menu select SVG.

- Recall the plane data contained in the data frame *Data_1*. This is a 45×2 table containing the plane number in one column and distance values in the second column. One way to plot this data is to display the distance means according to plane number side by side. The 95% confidence interval for each mean can also be shown.

1. First we must find the distance means of the three planes. To do this, we must first be able to call on the distances from the three planes individually. Recall how we accessed the second column:

```
> Data_1$distance
```

This gives the 45 distance values.

2. We want to pick out the distance values for plane number 1. We do this by:

```
> Data_1$distance[Data_1$plane==1]

 [1]  35  56  60  60  73  90 123 144 171 238 288 336 346 459 500
```

What this is saying is that we want all the distance values (*Data_1$distance*) when plane is equal to 1 (*Data_1$plane==1*). Assign this to *Plane_1*, and do a similar process to save the distance values for planes 2 and 3 to objects *Plane_2* and *Plane_3* respectively.

```
> Plane_1<- Data_1$distance[Data_1$plane==1]
> Plane_2<- Data_1$distance[Data_1$plane==2]
> Plane_3<- Data_1$distance[Data_1$plane==3]
```

Make sure you check that the contents of these three objects match those from the data frame. The easiest way to do this would be to first order the observations in the *Data_1* data frame according to plane, and then order them by distance within plane number. We can do both of these at the same time using the *order* function.

```
> Data_1[order(Data_1$plane, Data_1$distance), ]
```

This tells R that we want to order every row in the *Data_1* data frame firstly in terms of plane number, and then order the distances within each plane. This is done in an increasing fashion. Had we wanted to order them decreasingly, simply add a minus in front of the desired column within the *order* function. We can also reassign the ordered data frame to *Data_1*.

```
> Data_1<- Data_1[order(Data_1$plane, Data_1$distance), ]
```

3. Next we calculate the means and 95% confidence intervals for the three objects *Plane_1, Plane_2,* and *Plane_3.* The easiest way to do this is via the *t.test* function. The details of what this function tests will be formally covered later in the course, but for now we are only interested in the mean and confidence interval that are output from it.

```
> t.test(Plane_1)

        One Sample t-test

data:  Plane_1
t = 4.9997, df = 14, p-value = 0.0001946
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 113.4046 283.7954
sample estimates:
mean of x
    198.6
```

Record *mean of x* and *95 percent confidence interval* to three decimal places. Do the same for *Plane_2,* and *Plane_3.*

4. Create an object that contains the distance means for the three planes, and label this *Dist_Mean.*

```
> Dist_Mean<- c(198.600, 607.667, 488.067)
```

5. Create another object that contains the three lower limits of the 95% confidence intervals, and label this *Low_pt.*

```
> Low_pt<- c(113.405, 485.123, 387.559)
```

6. The final object contains the three upper limits of the 95% confidence intervals, and label this *Upp_pt.*

```
> Upp_pt<- c(283.795, 730.210, 588.574)
```

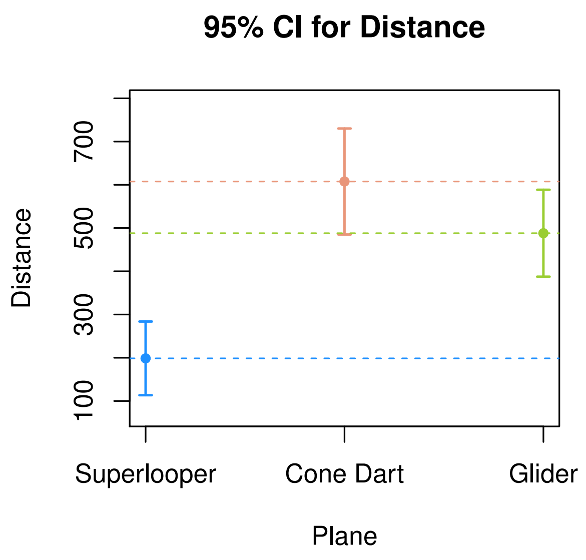
7. The plot we are going to create requires the *plotrix* package, load it in by:

```
> library(plotrix)
```

8. Next we use the *plotCI* function to plot the treatment means with corresponding CIs. Specify the number of planes in the first argument, followed by the means, upper, and lower limits respectively. The names of the three planes are Superlooper, Cone Dart, and Glider respectively.

```
> Number_Plane<- c(1,2,3)    # Specify the planes
> # Start the side-by-side CI plot
> plotCI(Number_Plane, Dist_Mean, ui= Upp_pt, li=Low_pt, lwd=1.5,
+        col=c("dodgerblue","darksalmon","olivedrab3"),
+        main= "95% CI for Distance", xlab= "Plane",
+        ylab= "Distance", ylim= c(70, 790),
+        pch=20,     # pch=20 gives solid dots for the plot symbols
+        xaxt= "n"   # xaxt="n" hides the x-axis scale
+        )
```

```
> # Set the x-axis scale to the actual names of the planes
> axis(side=1, at=1:3, labels=c("Superlooper","Cone Dart","Glider"))
> # Set reference lines across the mean points
> abline(h=Dist_Mean, lty=2,
+         col=c("dodgerblue","darksalmon","olivedrab3"))
```



Click the **Zoom** button under the **Plots** tab to view the plot with more detail. As mentioned, we use the *axis* command to specify the scale on the axis, in this case *side=1.* This refers to the x-axis. Had we wanted to change the *y*-axis, we would have used *yaxt="n"* and specified *side=2.*

To save a plot

(i) Follow the instructions given for saving the histogram.

1.12.2 Summary Statistics

Run some of the basic operations outlined in Section 1.6 on the objects *Sulphur_Data, Plane_1, Plane_2* and *Plane_3*.

1.13 Saving your work

1. (i) Save the R script by clicking on **File > Save**

2. (i) Save the **Workspace** so that you don't have to run your code when opening RStudio again.

   (ii) Click on the floppy disk icon under the **Workspace** tab.

   (iii) Specify the file name and set the folder to *Stat231_Intro_Lab*, click **Save**.

   (iv) After you have saved the workspace, clear it by clicking the broom icon to the right of the **Workspace** tab. The workspace goes blank. Now open the saved Workspace by clicking on the open icon to the left of the **Workspace** tab (brown folder with a green arrow coming out of it). Then find the file in the folder *Stat231_Intro_Lab.* 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.

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

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

```
# 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")
```

(iii) You can get to a new line in the console without running the code by holding $\boxed{\text{SHIFT}}$ and then hitting $\boxed{\text{ENTER}}$ at the same time.

(iv) Run this code.

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

(vi) 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.

1.14 References

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

2 Logbook Questions - Week 1

Each of the labs will be followed by *Logbook Questions.* A student will be required to attend at least 3 out of 4 labs to pass this subject and is required to finish all Logbook Questions (if not during lab, then at home). These will be checked after the fourth lab (week11). Print out your code, output and plots and all related work in a so-called **lab-book**. Lab 1 (week 1) should be followed by Lab 2 (week 4), etc. and it should be clear what is input and what output, etc.

Lab books are common practice in stats service subjects. In last years, only a small percentage of students came to labs and finished the exercises. To avoid this happening again, lab-books are now an assessment task and attendance is required.

Also note that some of the later assignments require some minor R programming. Hence it is essential to learn some basic R as quickly as possible, i.e. to complete labs exercises and lab-book questions.

You get 5 marks for each lab (20 in total), and all 4 labs together will be worth 5% of the total mark.

1. Create vector x, as the sequence of $0, 2, 4, 6, \ldots, 40$

2. Create vector y, defined as x^2

3. Store x and y in a data frame called, "data"

4. Plot y vs x using the plot command. Make sure the axes are properly labeled.

5. Save the data frame "data" as a csv-file "data.csv" using "write.table", use help(write.table) to see the options of write table, which is very similar to "read.table()"

6. Enter "rm(list=ls())" which deletes all objects

7. Enter "x" and show output

8. Load the data back into R with "read.table"

9. Calculate the sum of x and the (sample) mean/average of y