

# Statistics & Probability

## Statistical Programming with R

### Class 1 – Descriptive Statistics

#### **Why R?**

- Because it's free to download for everyone! Most statistical software is expensive, so this is a big advantage. Statisticians and users of statistics all over the world use R, so having some familiarity with it can be a big advantage. There are a number of advanced statistical models and techniques available for R which are not available for other packages. Once you are familiar with R, you can look at the code inside any function and customize it for your needs.

#### **Do I need any experience with programming for this?**

- No, the sheets you receive in each class will contain all the information you need. But you do have to pay attention to detail – commas, quotation marks, brackets and spelling are very important and are by far the most common sources of programming error in R.

#### **How will we be graded?**

- You will take a practical exam at the end of the semester which counts for 20% of your course credit. Lab attendance will be recorded and considered for students who are borderline between grades at the end of the course, so make sure to sign in each week.

#### **Can I get R on my own computer?**

- Yes, you can. Go to [r-project.org](http://r-project.org) and follow the download instructions. The UK “mirror” is generally a good source to download from (more on this in the download instructions). We will use the coding environment provided by RStudio in these classes. Once you have installed R, you can get Rstudio for free from [rstudio.com](http://rstudio.com).

#### **What if I have problems or questions?**

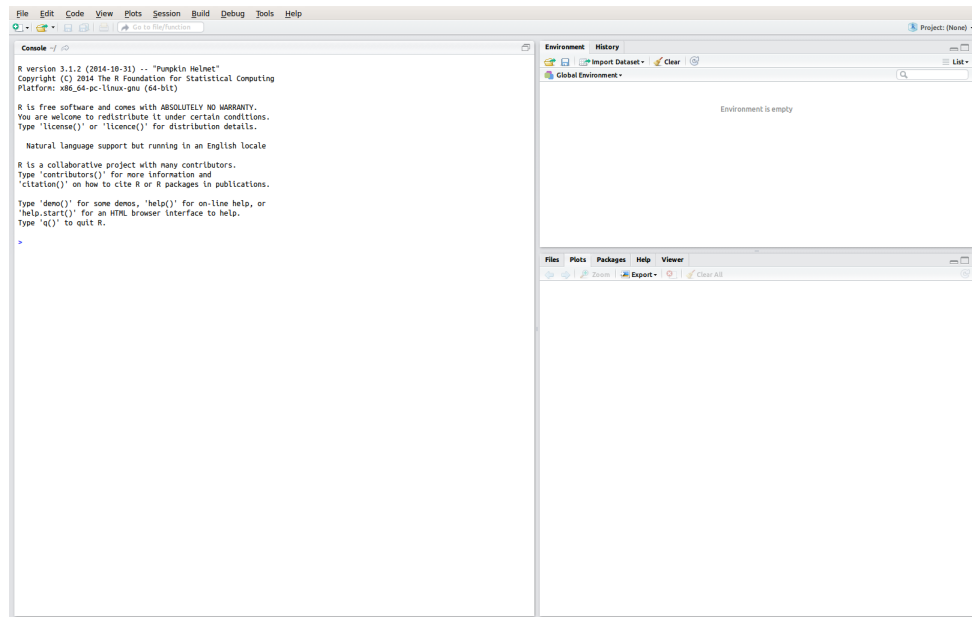
- Ask the instructors and/or work together on problems with your classmates. If you don't finish a lab sheet in your lab you are expected to complete it in your free time. Any problems you encounter can be dealt with by the instructors at your next lab.

# 1 Introduction

- To Open RStudio click:

Start → All Programs → Teaching Applications → RStudio

When **RStudio** opens, it will look like this:



- The simplest sort of object you will meet is a single number or string of letters. To create an object called **number1** with the value 7 type the following and press enter:

```
number1 <- 7
```

To see the value of the object called **number1**, type its name and press enter.

- You can make objects whose values are letters instead of numbers. To make an object called **word1** with the value “Word”, run:

```
word1 <- ‘‘Word’’
```

Notice that letters go inside quotation marks but numbers don’t.

- Suppose you want an object called **number2** whose value is two greater than **number1**. You have two choices. One is to type

```
number2 <- number1 + 2
```

This makes **R** find the value you already gave to **number1**, add 2 to it, and store the result as an object called **number2**.

- You can also use the **sum** function. Type

```
number2 <- sum(number1, 2)
```

This asks **R** to take everything inside the brackets, perform the **sum** function on them (i.e. add them up), and store the result as an object called **number2**. You can find more information on this function by typing

```
?sum  
or  
help(sum)
```

- **Note:** The help files are very important in **R**. There is also a function called **help.search()** that can be used to find an **R** function for a particular task, if you don't know the function's name. For example, type in

```
help.search('interquartile range')
```

You see that the function to compute it is called **IQR()**.

- While analyses can be carried out in **R** by typing commands directly into the terminal, a more useful approach is to open a script editor by clicking:

File → New File → R Script

This opens an internal editor which allows you to save your work as you go. To execute code from the editor simply highlight the relevant text, then press **Ctrl+Return**.

- In the terminal, up and down arrows can be used to cycle back and forth between previous commands typed. You can then edit the commands directly without having to completely rewrite the line of code if you make a mistake.
- If you want to make a note for yourself about what a piece of code does for future reference, use a “comment” in your separate script window – just type **#** before each line that is a comment. Even if this code is executed, **R** will not compile comments as a command. It is good practice to comment your code.

## 2 Data Types

### 2.1 Vectors

- We are not limited to working with one number or word at a time. Another type of object is a vector. A vector is just a list of words or numbers. If you want to make a vector containing the numbers 1, 3 and 6 for example, you use a function called `c()` which stands for concatenate. This function combines a list of objects into one vector. Type

```
vector1 <- c(1, 3, 6)
```

That is, take the three numbers in brackets (notice that they are separated by commas), turn them into a vector and save the result as an object called `vector1`. You can find out more about this function by looking up the help files. To see the vector, just type its name and press enter.

- You can use the `c` function on vectors as well as on numbers. If you want to add the values of `number1` and `number2` to the vector you just made, you can type

```
vector2 <- c(vector1, number1, number2)
```

This takes `vector1` (which is of length three), `number1` and `number2` and puts them all together in `vector2` which is of length five. Type `vector2` to see your new vector.

- What if you want a vector with words instead of numbers? The `c` function can do that as well. Type

```
vector3 <- c('one', 'three', 'six', 'seven', 'nine')
```

This is just the same as making a vector with numbers in it, except that the words in the brackets are in quotation marks. Type `vector3` to make sure that it worked.

### 2.2 Data Frames

- The third type of object we will use is called a data frame. A data frame is like a matrix or a table. You can make one by putting two or more vectors together. The function to do this is called `data.frame`.

```
data1 <- data.frame(vector2, vector3)
```

makes an object called `data1` which contains both vectors as columns. Type `data1` to see what it looks like. Notice that the columns are headed with the names of the vectors, and the rows are numbered. You can find more information on this function by typing

```
?help(data.frame)
```

- The key difference between a data frame and a matrix is that the columns in a data frame can contain variables of different types. Now we have some basic functions to work with, let's deal with an example.
- A teacher has the following data available on five students:

Name	Exam 1	Exam 2
John	92	82
Anne	75	96
Terry	98	60
Fred	62	55
Maria	79	72

- We will begin by making a table in **R** with this data. To do that, we first make a vector in **R** for each of the three variables. At the prompt, type

```
studentname <- c('John', 'Anne', 'Terry', 'Fred', 'Maria')
```

and press enter. Notice that there are commas in between each name. The names are in quotation marks because they are made of letters, not numbers. This makes a single vector of names in **R**. The vector is called `studentname`. To check that it has worked, type `studentname`. You should see the list of names printed in the window.

- Now we will do the same thing for Exam 1 and Exam 2:

```
exam1 <- c(92, 75, 98, 62, 79)
```

Because these are numbers, there are no quotation marks.

- Make a third list of numbers called `exam2` containing the marks for Exam 2.

**\*Complete part A on your answer sheet\***

- Now, we want to make one table with all this information in it. Type

```
results <- data.frame(studentname, exam1, exam2)
```

This makes a single table (which **R** calls a “dataframe”) with three columns and five rows. To see what it looks like, type `results`.

- Next, the teacher wants to find the average of the two exams for each student. So we need to make a fourth column in the `results` dataset containing this value for each student. Type:

```
results$avg <- ((results$exam1 + results$exam2)/2)
```

- Notice how we name columns by typing the name of the dataset, then a “\$” sign, then the name of the column.
- Next, we will assign letter grades to the students. The bands are as follows:

< 50	D
50 – 69	C
70 – 84	B
85 – 100	A

- Type:

```
results$grade <- cut(results$avg, breaks = c(0, 49, 69, 84, 100), labels = c("D", "C", "B", "A"))
```

Notice how we are still naming columns with the `dataset$column_name` method, and how the letters are inside quotation marks but the numbers are not.

**\*Complete part B on your answer sheet\***

- Next, we want to sort the data from highest mark to lowest mark. Type:

```
results <- results[order(results$avg, decreasing=TRUE),]
```

What is going on here? Well, we are asking **R** to sort all of the data in `results` in order of the data in the column called `average`. Since **R** sorts in ascending order by default, we asked it to sort in the opposite order by adding `decreasing=TRUE` to the inputs to `order`.

- Lastly, suppose we want to print only the students' names and letter grades. First, we make a new dataframe with only those two columns in it. Type

```
results2<-data.frame(results$studentname, results$grade)
```

And then print it to screen by typing `results2`.

**\*Complete part C on your answer sheet\***

### 3 Importing Data

- In the previous section, we typed data directly into **R**. For large datasets, that would get very tedious. More often, you will have data which is already saved in your computer. To import text files, we use the `read.table` function.
- Download the dataset called `pulse.txt` from blackboard and save it into your documents folder.
- For convenience we will make this folder your 'working directory', i.e. this is where R will look for files you wish to import. In the bottom right pane on RStudio, click on the **Files** tab, then click on the button with the 3 dots on the right hand side. A new window will pop up. In this window, click on the folder which has your name then on MyDocuments and click OK. Finally click

More → Set as Working Directory

- Open a new **R** script and type
 

```
pulse <- read.table('pulse.txt', header = TRUE)
pulse <- data.frame(pulse)
```
- To read in .csv files, follow the same steps but use the function `read.csv`.

### 4 Descriptive Statistics

- The first two columns of `pulse` contain the resting pulse and pulse after exercise of 92 volunteers. The third contains 1 or 2 depending on the type of exercise the person took – walking or running. The fourth column contains the sex of the person (1 for males and 2 for females). The last column contains the volunteer's weight.
- Unfortunately a data entry error means we do not know which of the first two columns is resting pulse and which is pulse after exercise. In the third column, we

don't know whether 1 or 2 means the person ran instead of walking. We're going to use the data to find out these things.

- A useful function in **R** is called **summary**. It is performed on a single column or a dataframe. If used with a single column, it returns the mean (average), median (middle value), minimum (smallest value) and maximum (biggest value) in the column. If used with a dataframe, it does this for each column in it.
- To use it for the column **pulse1** within the dataframe **pulse**, type

```
summary(pulse$pulse1)
```

To use it on the whole dataframe, type

```
summary(pulse)
```

You can find more information about `summary` by typing `?summary`.

- The four numbers returned by **summary** can also be obtained individually using the `mean()`, `median()`, `min()` and `max()` functions respectively.

### **\*Complete Part D on your answer sheet\***

- Next, we want to know whether 1 or 2 in the **ran** column means the person ran. To do that, we first need to separate out the rows where **ran** = 1 from the rows where **ran** = 2.
- Luckily, we have a quick way of telling R to apply a function to some, not all, of a given column. The general shape is

```
function(column[condition])
```

That is, apply a certain function to a certain column, but only for the part of the column where a certain logical condition is true. Here, the function is **summary**. The column we want to apply it to is **pulse\$pulse2**. But we only want to apply it to cases where **pulse\$ran** = 1. The syntax to specify that condition is

```
pulse$ran == 1 (with 2 “=” signs!)
```

So the whole thing is put together as



```
summary(pulse$pulse2[pulse$ran == 1])
```

- Now do the same for the people who have 2 in the ran column.

**\*Complete part E on your answer sheet\***

## 5 Graphs

- Often, it is easier to see what is going on by looking at a graph. One type of graph you may have seen in lectures is a histogram. To draw one, use the `hist` function. This function takes a single column of data and draws a histogram of it. There are other options too – you can find out more by typing `?hist`.
- Create a histogram of `pulse1` by typing

```
hist(pulse$pulse1)
```

- Now suppose we want a histogram of `pulse2`. Type

```
hist(pulse$pulse2)
```

A new histogram appears – and the previous histogram disappears! This is why you should always copy your graphs into another document. Click:

File → Copy as Bitmap → Paste

Simply use a word document as the place to copy them to. Comparing the histograms for `pulse1` and `pulse2`, which would you say is the pulse after exercising?

- It's important to label and title graphs accurately. There are two more options in the `hist` function - `xlab`, which gives a label to the x-axis (`ylab` for the y-axis), and `main`, which gives a title to the whole graph. You use them like this:

```
hist(pulse$pulse2, xlab = "Pulse", main = "Histogram of Pulse Rate")
```

- Draw a histogram of the weights of female volunteers, with appropriate labels and title.

**\*Complete part F on your answer sheet\***

- We also looked at other graphs such as bar charts and box plots in lectures. These are easily plotted in **R**.

- To produce a bar chart summarising the sex of the volunteers in this study type.

```
barplot(table(pulse$sex))
```

- The `table` function returns a frequency table for the column `sex`. Run this function by itself to see what it returns. Give the plot an appropriate title and label the axes by adding extra inputs to the `barplot` function as we did for the `hist`. For more information on this function look up the help files.
- The histograms we produced earlier allowed us to compare the spread and location of `pulse1` to that of `pulse2`. We can look at this comparison in more detail using box plots. To produce box plots of the columns `pulse1` and `pulse2` type

```
boxplot(pulse$pulse1, pulse$pulse2, names=colnames(pulse)[1:2])
```

Any observations beyond the whiskers of the box plot are far from the bulk of the data and could be outliers. Give the plot an appropriate title in the same way as before. Using the help files can you work out what `names=colnames(pulse)[1:2]` does to the plot?

**\*Complete part G on your answer sheet\***

## 6 Other Data Sets

- We looked at a number of datasets in class. Some of these are available as part of R and can be used to practice what you learnt in class.

```
data(morley) # loads the Michelson-Morley data
data(faithful) # loads the Old Faithful data
```

- In the `morley` data object there are a number of columns that contain the measurements but also the experiment number. If you compute the summary statistics for each experiment, are the results similar or different for each experiment?

**Make sure you save your R script to a USB key or to Google drive before you log off.**

# Statistics & Probability

## 1 – Introduction and Descriptive Statistics

A. What code creates the list of `exam2` marks?

---



---



---



---

B. Who has a better grade, Maria or Terry?

---



---

C. Suppose you wanted to print out only the names and Exam 1 marks of the students. What code would you type?

---



---

D. Pulse 1 and Pulse 2:

pulse1	pulse2
Mean:	Mean:
Median:	Median:

Which do you think is pulse after exercise?

---



---

E. Ran:

ran = 1	ran = 2
Mean:	Mean:
Median:	Median:

Which group do you think ran and which walked?

---

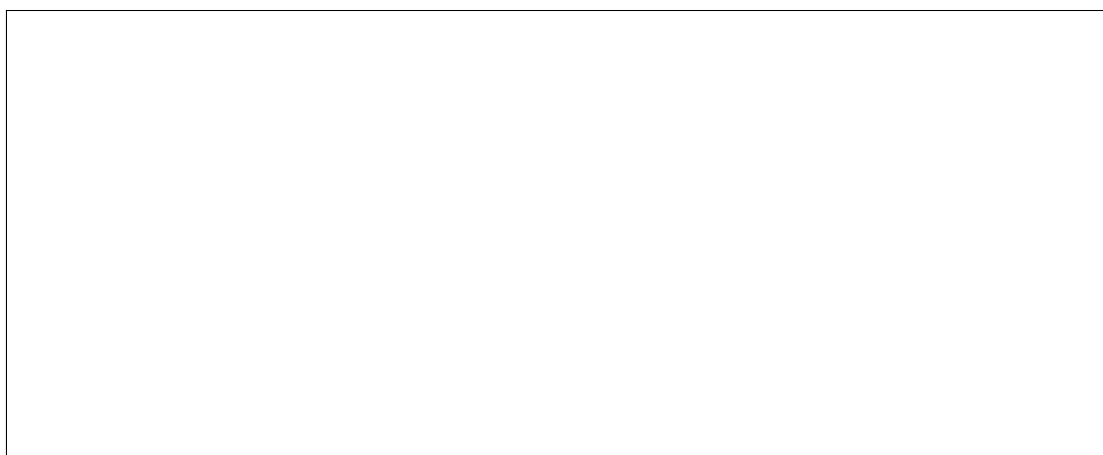
---

What code did you use to calculate this?

---

---

F. Sketch the histogram of the weights of female volunteers here:



What code did you use to produce this histogram?

---

---

G. Which variable appears to have a larger spread, `pulse1` or `pulse2`? Explain.

---

---

Using the box plots, give a rough estimate of the interquartile range for `pulse1` and `pulse2`.

<code>pulse1</code>	<code>pulse2</code>
IQR: _____	IQR: _____