Macquarie R Users Group - An Introduction to R (written in R Notebook (Markdown))

Link to the presentation

Prerequisite

Install R and R Studio according to the manual provided in the "R Users Group Beginner Session V1"-file. If you have it already installed pls make sure you have the latest version installed.

Downloading and Installing R and R studio (IDE)

1. Download R:

a. For Windows just click: https://cran.r-project.org/bin/windows/base/R-3.6.0-win.exe
b. If you use another OS: i. Go to: https://www.r-project.org/. ii. On the left-hand side panel click 'CRAN' and choose a preferred download mirror (you can also just click: https://cran.ms.unimelb.edu.au/). iii. Follow the instructions on the webpage and download R for your preferred OS; choose 'install R for the first time.' and 'Download R 3.6.0 for Windows'.

2. Install and execute R.

3. Download R Studio here. Just pick the desired installer for your OS.

4. Install R studio and execute.

5. Be familiar with the expression IDE (find link in title above).

6. Click: http://www.r-project.org

7. On the left-hand side panel click 'CRAN' and choose a preferred download mirror (you can also just click 'CRAN' here)

8. Download R for your preferred OS (for Windows just click here)

9. If you use another OS just follow the instructions on the website and choose 'install R for the first time.' and 'Download R 3.6.0 for Windows'

Section 1

- Sit together in pairs. Envision yourself as navigator and programmer. Swap tasks once in a while.

Goals

1. Getting comfortable with R Studio Interface and finding out what it is at all.
2. Using basic commands.
3. Loading and saving data.
4. Basic statistics.
5. Plotting.
6. Not being scared of coding!

What is R?

"R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS." (CRAN)

"In its broadest definition, R is a computer language that allows the user to program algorithms and use tools that have been programmed by others." (Zuur et al 2009 - A Beginner's Guide to R:14)

But what can it actually do?

- R as a calculator
- Manipulate data
- Conduct any statistical test
- Import software 'packages' with specialised functions (more on this later)
- Automate analyses
- Design simple or complicated graphs

Why you should use it?

- It is free and open-source
- R has been receiving contributions from many programmers around the globe
- Listed as 3rd most used languages in Data Science
- Massive community for online support
- Very specific problems are mostly addressed with a package
- It is widely used with many books published in the last years

Awesome! Why is not everyone using it?

- A bit of a learning curve
- Coding necessary (eh!)

** BUT **

- The most basic syntax (grammar) can be used for most of the things in R
- R studio makes it easier to code in R, providing a user friendly interface
- Once you get used to programming you can adopt new languages easier

There are a lot of online courses, videos and texts available for understanding R, its packages etc.

Let's have a look at R Studio

What's what:

- Console: your code is run here and you will see the results of your coding.
- R-Script: your code is written and saved here, just like in a normal text-document.
- Environment: all the loaded data and objects are listed here, you can even take a look at your data tables or the structure of your data.
- History: shows the history of your executed code.
- Files: what is in your source folder, i.e. is my data table in the folder?
- Plots: this is where your plots will be shown, you can also export them from here (but there are better ways).
- Packages: load and search for new packages and your installed packages are listed here.
- Help: look for help or specific vignettes (support documents) for each package. Also access via ¿function'
- Viewer: can be used to view local web content for web graphics generated using packages like googleVis, htmlwidgets, and rCharts, or even a local web application created using Shiny, Rook, or OpenCPU.

R-studio gives you a more intuitive interface and takes the scariness out of coding. It also provides functions that simplify the process of developing your code. 'Tab completion' is one of such function.

Some basic R syntax: objects and functions and arguments

```
# output<-function1(argument1, argument2, ...) + function2(argument)

# flat_white <- froth(milk, hot) + extract(coffee)
# flat_white = froth(milk, hot) + extract(coffee)

# verb(argument)
# argument can be a "noun" (being acted upon) or an "adverb" (modifying its behavior)

# Example1: boiling milk normally
# boiled_milk <- boil (milk)

# Example2: boiling milk for a long time
# boiled_milk <- boil (milk, long_time)
```

froth() and extract() are *functions*, milk, hot and coffee are *arguments*

functions are sets of instructions used to do something to arguments. They can be stored in an *object* (flat_white). Objects can be used as arguments.

arguments are used to tell functions what *objects* to act on, and any details of how to perform the action

Functions need arguments to fulfill the purpose they were designed for. e.g. froth() needs to know what kind of milk to froth and how hot to make it.

packages are precoded sets of instructions (functions) that were written by someone and are available for everyone to use

Now it's time to play around in R. We will create some dummy data and create a basic scatterplot.

1. We can assign (<-) a basic calculation to the object 'a' and call the content of 'a'. Execute your code using Ctrl+Enter

```
a <- 1+2 # here R works like a calculator
a        # print a to see what it contains
```

```
## [1] 3
```

2. We Use function c() to combine specific values into a vector. Assign this new vector to object 'x'.

```
x <- c(1,2,3,4) # 'c' is a function that combines values into the vector x (object), the numbers are ar
```

A vector is a sequence of data *components* of the same basic type (i.e. numbers or letters)

3. Using function mean (), we can extract the mean of our vector.

```
mean(x) # mean() is a function
```

```
## [1] 2.5
```

4. Create two vectors (they are going to be numeric in our case) using seq() and assign them the object seq_a and seq_b. object seq_a contains a vector with the components 1 to 10 and is increasing by 1. seq_b contains the components 1 to 25 and increases by 2. If you are not sure how to use a function, such as seq(), just call ?seq and have a look what arguments can be used.

```
seq_a <- seq(from=1,to=10,by=1)
seq_b <- seq(from=1,to=25,by=2)
```

5. Using cbind() you can bind two vectors to create a *matrix* (a kind of table). Use cbind() to bind seq_a and seq_b. Assign it to the object 'c'. For help, call ?cbind

```r
# uncomment this next line and pay attention to what happens when this like is run

# c <- cbind(seq_a,seq_b)
```

6. Oops! Let's see what went wrong. Can you decipher the error message? Have a look at seq_a and seq_b. Just type seq_a and seq_b and execute both. No worries, debugging (resolving errors) is a major part of programming.

```r
seq_a
```

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

```r
seq_b
```

```
##  [1]  1  3  5  7  9 11 13 15 17 19 21 23 25
```

7. To make the problem more obvious, let's check the length of each object. Use length().

```r
length(seq_a)
```

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

```r
length(seq_b)
```

```
## [1] 13
```

8. To cbind() two vectors they have to have the same length. Let's overwrite seq_a and create a vector of the same length as seq_b. Check if the length is matching the other vector and bind them using cbind(). Assign this object to a new object, 'c'. What class has 'c'? Check it!

```r
seq_a<- seq(1,13,by=1)

length(seq_a)
```

```
## [1] 13
```

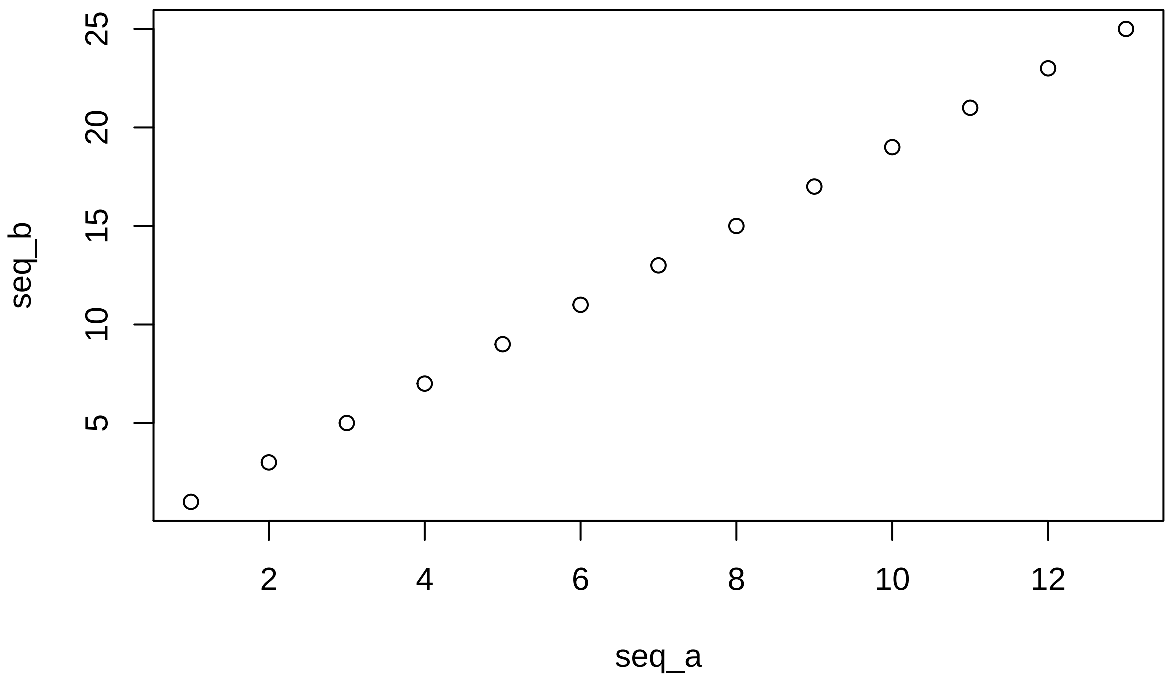
```r
c <- cbind(seq_a,seq_b)

class(c) #class() can figure out if you are working with vectors, matrices, dataframes, lists etc....it
```

```
## [1] "matrix"
```

**Note: The counterpart to cbind() is rbind() if you would like to connect rows instead of columns.

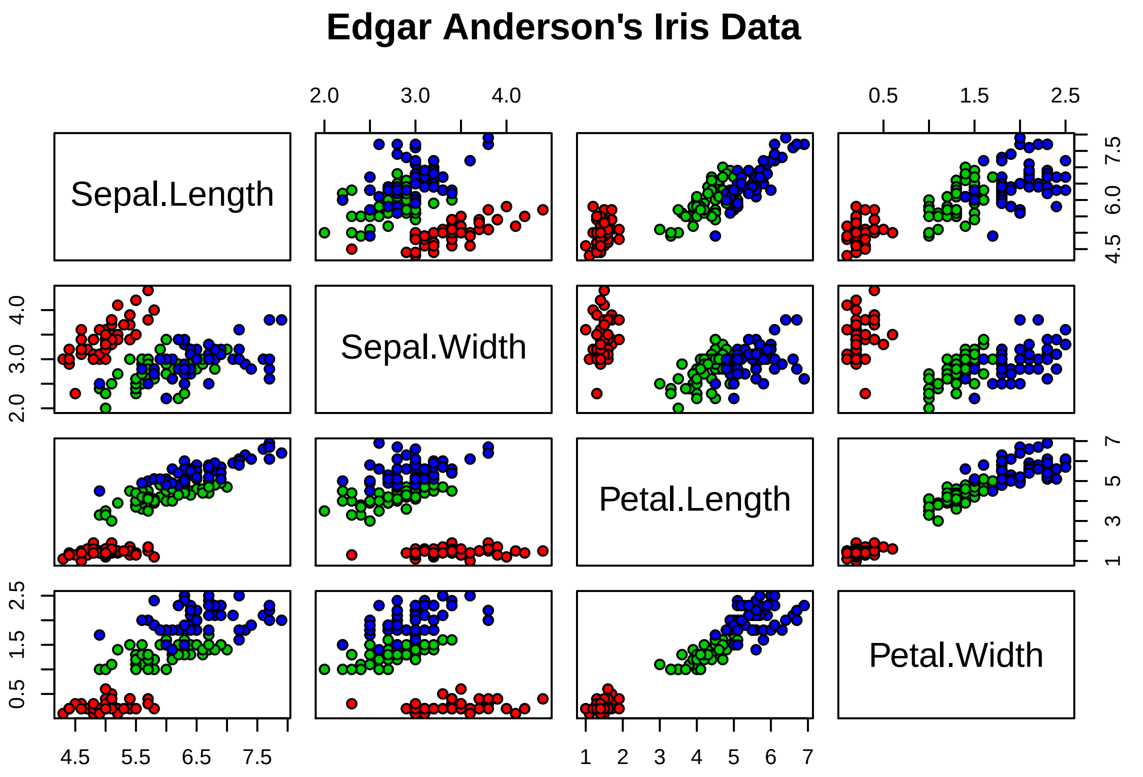
9. Plot 'c' by using plot()

```r
plot(c)
```



Note:
If we wanted to, we could modify the appearance of this plot completely. Labelling axis, change tickmarks and intervals, add text or shapes... more than you can think of now. With just a few lines of code we can create beautiful plots. Once a plot is coded we can use it over and over again and also easily modify it. See here:

```
pairs(iris[1:4], main = "Edgar Anderson's Iris Data", pch = 21, bg = c("red","green3","blue")[unclass(i
```



###
What we have learned:

- Get an idea of what R can possibly do

- Discover R Studio
- Become familiar with some basic expressions
- Encounter error messages
- Create some first data
- Have an idea that there are different classes that R can use (different packages want different classes)
- See what a basic plot looks like and how it could look like (Know that there are different ways/ packages of plotting something)

Section 2

Getting Data into R

How do we get started?

- Create a new project folder for our R users introduction course: R project > New project > New directory > Browse and name it: 'My first R project'.
- Create 3 subfolders within the project and name them 'Input', 'Output' and 'Scripts'
- Move our data to input folder
- Create new script: File > New File > R script
- Start coding

1. Let's import our data and see what it looks like

```
# if the dataset is build in R, it is unnecessary to export it as csv and import it, you just need the
# it is the case with iris and PlantGrowth datasets, so they can be loaded using:
data(iris)
data(PlantGrowth)

# or

irisdata <- read.csv("Input/irisdata.csv")
irisdata
```

```
##      X Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1    1          5.1         3.5          1.4         0.2  setosa
## 2    2          4.9         3.0          1.4         0.2  setosa
## 3    3          4.7         3.2          1.3         0.2  setosa
## 4    4          4.6         3.1          1.5         0.2  setosa
## 5    5          5.0         3.6          1.4         0.2  setosa
## 6    6          5.4         3.9          1.7         0.4  setosa
## 7    7          4.6         3.4          1.4         0.3  setosa
## 8    8          5.0         3.4          1.5         0.2  setosa
## 9    9          4.4         2.9          1.4         0.2  setosa
## 10  10          4.9         3.1          1.5         0.1  setosa
## 11  11          5.4         3.7          1.5         0.2  setosa
## 12  12          4.8         3.4          1.6         0.2  setosa
## 13  13          4.8         3.0          1.4         0.1  setosa
## 14  14          4.3         3.0          1.1         0.1  setosa
## 15  15          5.8         4.0          1.2         0.2  setosa
## 16  16          5.7         4.4          1.5         0.4  setosa
##  [ reached 'max' / getOption("max.print") -- omitted 134 rows ]
```

```
# Why using .csv instead of Excel sheets (.xls and .xlsx)?
```

2. We can easily call some summary stats now.

```r
summary(irisdata)
```

```
##        X          Sepal.Length    Sepal.Width     Petal.Length  
##  Min.   :  1.00   Min.   :4.300   Min.   :2.000   Min.   :1.000  
##  1st Qu.: 38.25   1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600  
##  Median : 75.50   Median :5.800   Median :3.000   Median :4.350  
##  Mean   : 75.50   Mean   :5.843   Mean   :3.057   Mean   :3.758  
##  3rd Qu.:112.75   3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100  
##  Max.   :150.00   Max.   :7.900   Max.   :4.400   Max.   :6.900  
##   Petal.Width          Species  
##  Min.   :0.100   setosa    :50  
##  1st Qu.:0.300   versicolor:50  
##  Median :1.300   virginica :50  
##  Mean   :1.199                  
##  3rd Qu.:1.800                  
##  Max.   :2.500                  
```

3. We can also access specific values in this dataset. For vectors, matrices and dataframes we can use "[]", and the "$" is useful only for dataframes. If we use "[]" then we must think of it like this: [rows,columns]

```r
irisdata[,1] # all values in column 1
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
##  [17]  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32
##  [33]  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48
##  [49]  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64
##  [65]  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
##  [81]  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96
##  [97]  97  98  99 100
##  [ reached getOption("max.print") -- omitted 50 entries ]
```

```r
irisdata[1,1] # value at row 1, column 1
```

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

```r
irisdata[,1:3] # all values in columns 1 to 3
```

```
##     X Sepal.Length Sepal.Width
## 1   1          5.1         3.5
## 2   2          4.9         3.0
## 3   3          4.7         3.2
## 4   4          4.6         3.1
## 5   5          5.0         3.6
## 6   6          5.4         3.9
## 7   7          4.6         3.4
## 8   8          5.0         3.4
## 9   9          4.4         2.9
## 10 10          4.9         3.1
## 11 11          5.4         3.7
## 12 12          4.8         3.4
## 13 13          4.8         3.0
## 14 14          4.3         3.0
## 15 15          5.8         4.0
## 16 16          5.7         4.4
## 17 17          5.4         3.9
## 18 18          5.1         3.5
## 19 19          5.7         3.8
## 20 20          5.1         3.8
## 21 21          5.4         3.4
## 22 22          5.1         3.7
## 23 23          4.6         3.6
## 24 24          5.1         3.3
## 25 25          4.8         3.4
## 26 26          5.0         3.0
## 27 27          5.0         3.4
## 28 28          5.2         3.5
## 29 29          5.2         3.4
## 30 30          4.7         3.2
## 31 31          4.8         3.1
## 32 32          5.4         3.4
## 33 33          5.2         4.1
##  [ reached 'max' / getOption("max.print") -- omitted 117 rows ]
```

```
irisdata['Species'] # all values in column with column name 'Species'
```

```
##        Species
## 1       setosa
## 2       setosa
## 3       setosa
## 4       setosa
## 5       setosa
## 6       setosa
## 7       setosa
## 8       setosa
## 9       setosa
## 10      setosa
## 11      setosa
## 12      setosa
## 13      setosa
## 14      setosa
## 15      setosa
## 16      setosa
## 17      setosa
## 18      setosa
## 19      setosa
## 20      setosa
## 21      setosa
## 22      setosa
## 23      setosa
## 24      setosa
## 25      setosa
## 26      setosa
## 27      setosa
## 28      setosa
## 29      setosa
## 30      setosa
## 31      setosa
## 32      setosa
## 33      setosa
## 34      setosa
## 35      setosa
## 36      setosa
## 37      setosa
## 38      setosa
## 39      setosa
## 40      setosa
## 41      setosa
## 42      setosa
## 43      setosa
## 44      setosa
## 45      setosa
## 46      setosa
## 47      setosa
## 48      setosa
## 49      setosa
## 50      setosa
## 51  versicolor
## 52  versicolor
## 53  versicolor
## 54  versicolor
## 55  versicolor
## 56  versicolor
## 57  versicolor
## 58  versicolor
## 59  versicolor
## 60  versicolor
## 61  versicolor
## 62  versicolor
## 63  versicolor
## 64  versicolor
## 65  versicolor
## 66  versicolor
## 67  versicolor
## 68  versicolor
## 69  versicolor
## 70  versicolor
## 71  versicolor
## 72  versicolor
## 73  versicolor
## 74  versicolor
## 75  versicolor
## 76  versicolor
## 77  versicolor
## 78  versicolor
## 79  versicolor
## 80  versicolor
## 81  versicolor
## 82  versicolor
## 83  versicolor
## 84  versicolor
## 85  versicolor
## 86  versicolor
## 87  versicolor
## 88  versicolor
## 89  versicolor
## 90  versicolor
## 91  versicolor
## 92  versicolor
## 93  versicolor
## 94  versicolor
## 95  versicolor
## 96  versicolor
## 97  versicolor
## 98  versicolor
## 99  versicolor
## 100 versicolor
##  [ reached 'max' / getOption("max.print") -- omitted 50 rows ]
```

```
irisdata$Sepal.Length # all values in column with column name 'Sepal.length'
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7
##  [17] 5.4 5.1 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4
##  [33] 5.2 5.5 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6
##  [49] 5.3 5.0 7.0 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1
##  [65] 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7
##  [81] 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7
##  [97] 5.7 6.2 5.1 5.7
##  [ reached getOption("max.print") -- omitted 50 entries ]
```

```
#as.matrix(irisdata)$Sepal.Length

# this won't work. atomic vectors = (logical, integer, double (sometimes called numeric), and character

# this will work
as.matrix(irisdata)
```

```
##        X     Sepal.Length Sepal.Width Petal.Length Petal.Width
##   [1,] "  1" "5.1"        "3.5"       "1.4"        "0.2"
##   [2,] "  2" "4.9"        "3.0"       "1.4"        "0.2"
##   [3,] "  3" "4.7"        "3.2"       "1.3"        "0.2"
##   [4,] "  4" "4.6"        "3.1"       "1.5"        "0.2"
##   [5,] "  5" "5.0"        "3.6"       "1.4"        "0.2"
##   [6,] "  6" "5.4"        "3.9"       "1.7"        "0.4"
##   [7,] "  7" "4.6"        "3.4"       "1.4"        "0.3"
##   [8,] "  8" "5.0"        "3.4"       "1.5"        "0.2"
##   [9,] "  9" "4.4"        "2.9"       "1.4"        "0.2"
##  [10,] " 10" "4.9"        "3.1"       "1.5"        "0.1"
##  [11,] " 11" "5.4"        "3.7"       "1.5"        "0.2"
##  [12,] " 12" "4.8"        "3.4"       "1.6"        "0.2"
##  [13,] " 13" "4.8"        "3.0"       "1.4"        "0.1"
##  [14,] " 14" "4.3"        "3.0"       "1.1"        "0.1"
##  [15,] " 15" "5.8"        "4.0"       "1.2"        "0.2"
##  [16,] " 16" "5.7"        "4.4"       "1.5"        "0.4"
##        Species
##   [1,] "setosa"
##   [2,] "setosa"
##   [3,] "setosa"
##   [4,] "setosa"
##   [5,] "setosa"
##   [6,] "setosa"
##   [7,] "setosa"
##   [8,] "setosa"
##   [9,] "setosa"
##  [10,] "setosa"
##  [11,] "setosa"
##  [12,] "setosa"
##  [13,] "setosa"
##  [14,] "setosa"
##  [15,] "setosa"
##  [16,] "setosa"
##  [ reached getOption("max.print") -- omitted 134 rows ]
```

```r
# why won't this run?

# irisdata[1, 1:7] # first row only of values in columns 1 to 7

# fixed

irisdata[1, 1:6]
```

```
##   X Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1 1          5.1         3.5          1.4         0.2  setosa
```

```r
dim(irisdata) #shows dimensions
```

```
## [1] 150   6
```

4. If we make any changes to our data, we can save our new data in a spreadsheet.

```r
write.csv(irisdata, 'New irisdata.csv', row.names=FALSE) # Why am I using row.names=FALSE?
write.csv(irisdata, 'New irisdata incl rownames.csv')
```

Nice! We have learned a lot about manipulating data so far! Use R cheat sheets (just google R cheatsheets) to look up all those functions over and over again!

Last part! Our first data analysis!

1. Read in a new dataset in csv

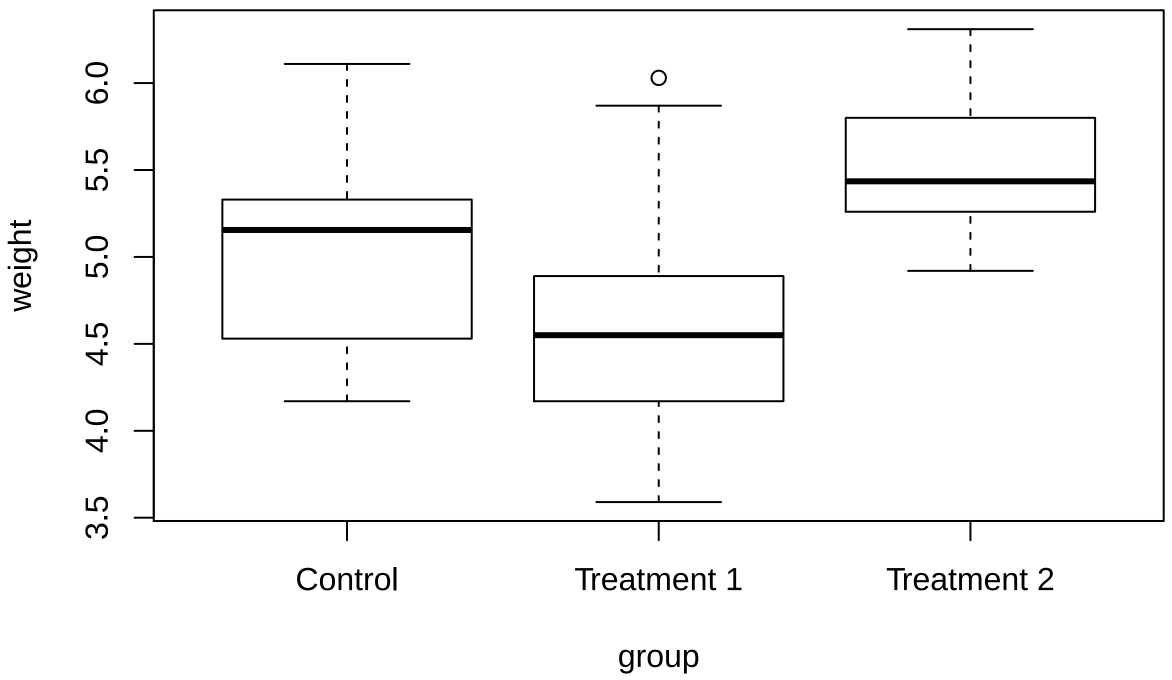
```r
plant.df <- read.csv("Input/PlantGrowth.csv")
```

2. Clean the data up a bit and specify that the group is a factor variable.

```r
plant.df$group <- factor(plant.df$group,
  labels = c("Control", "Treatment 1", "Treatment 2"))
```

3. Visualise our data with a boxplot.

```r
boxplot(weight~group, plant.df)
```



4. Create a folder to store the results.

```
# this line can be different for Mac users
dir.create("output")
```

```
## Warning in dir.create("output"): 'output' already exists
```

And save it as a .pdf file in the output folder.

```
pdf('output/My Boxplot.pdf', width = 20, height = 10 , paper = 'a4r')
boxplot(weight~group, plant.df, ylab='Dried weight of plants [g]')
dev.off()
```

```
## pdf
##   2
```

4. Start statistical analysis. This is a simple linear model with an ANOVA.

```
plant.mod1 = lm(weight ~ group, data = plant.df) # we're using lm() to create a pretty different object
summary(plant.mod1) # summary() extracts some of this data and prints it out neatly for us
```

```
##
## Call:
## lm(formula = weight ~ group, data = plant.df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.0710 -0.4180 -0.0060  0.2627  1.3690
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)           5.0320     0.1971  25.527   <2e-16 ***
## groupTreatment 1     -0.3710     0.2788  -1.331   0.1944
## groupTreatment 2      0.4940     0.2788   1.772   0.0877 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 0.6234 on 27 degrees of freedom
## Multiple R-squared:  0.2641, Adjusted R-squared:  0.2096
## F-statistic: 4.846 on 2 and 27 DF,  p-value: 0.01591
```

```
anova(plant.mod1)
```

```
## Analysis of Variance Table
##
## Response: weight
##           Df  Sum Sq Mean Sq F value  Pr(>F)
## group      2  3.7663  1.8832  4.8461 0.01591 *
## Residuals 27 10.4921  0.3886
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

5. There are hundreds of packages in R that have ready functions for us to use. All you need to do is look up which package you need, install it and load it into R. In our case, as we ran a linear model, we probably want to visualise our model estimates instead of just using a boxplot. Let us install and load a new package that allows to easily do this.

```
# function to use an improved read.csv function

#install.packages('readr') #install
library(readr) #load
```

Now all we have to do is use a function within the newly loaded package!

```
irisdata <- read_csv("Input/irisdata.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   X1 = col_double(),
##   Sepal.Length = col_double(),
##   Sepal.Width = col_double(),
##   Petal.Length = col_double(),
##   Petal.Width = col_double(),
##   Species = col_character()
## )
```

```
irisdata
```

```
## # A tibble: 150 x 6
##       X1 Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##    <dbl>        <dbl>       <dbl>        <dbl>       <dbl> <chr>
##  1     1          5.1         3.5          1.4         0.2 setosa
##  2     2          4.9         3            1.4         0.2 setosa
##  3     3          4.7         3.2          1.3         0.2 setosa
##  4     4          4.6         3.1          1.5         0.2 setosa
##  5     5          5           3.6          1.4         0.2 setosa
##  6     6          5.4         3.9          1.7         0.4 setosa
##  7     7          4.6         3.4          1.4         0.3 setosa
##  8     8          5           3.4          1.5         0.2 setosa
##  9     9          4.4         2.9          1.4         0.2 setosa
## 10    10          4.9         3.1          1.5         0.1 setosa
## # ... with 140 more rows
```

More Information

Resources to learn R coding * Book A Beginner's Guide to R (Use R!) - Alain Zuur, Elena Ieno and Eric Meesters * Package (Swirl)

Resources to learn plotting with R Base Graphics * R Graph Cookbook - Hrishi V. Mittal

Resources to learn plotting with ggplot2 * ggplot2 (Use R!) - Hadley Wickham

Resources to learn data manipulation in R * Data manipulation with R (Use R!) - Phil Spector

Resources to learn stats in R * Introductory statistics with R (Use R!) - Peter Dalgaard

What we have learned

- Get familiar with R Studio and the differences to R
- How to import and export data in R?
- What do projects and setwd() have in common and what is its purpose?
- How to manipulate data?
- Your first data analysis
- How to proceed on your own