# INTRODUCTION TO R

Lincoln University LTL PG workshop

A basic introduction to the R statistics packages and programming environment

October 2019

Material prepared by Alexa Byers
Alexa.byers@lincolnuni.ac.nz

October 2019

## Contents

What is R? .... 2
Getting started with R .... 2
    The R interface .... 2
Data types and structures .... 4
    Types .... 4
    Structures .... 4
    Vectors .... 4
    Lists .... 6
    Matrices .... 6
    Data Frames .... 6
    Factors .... 7
Using R studio .... 7
    Obtaining help for a command .... 8
Importing data in R Studio .... 9
    Setting up your working directory or folder .... 9
    Importing data from Excel .... 9
Handling data .... 10
    Summarising data .... 10
    Plotting data .... 11
    Reshaping data .... 12
    Summarising continuous data .... 14
Exporting data from R studio .... 15
    Workspace and history .... 15
    Data export .... 15
R scripts .... 15
Basic statistics .... 16
    t-test .... 16
    ANOVA via a simple linear model .... 16
    ANOVA and multiple comparison tests .... 17
    Simple linear regression .... 18
Useful R functions and packages .... 20
    Community resources .... 23

## What is R?

R is a free, open source statistical package widely used in academia and amongst environmental scientists and ecologists. It functions as both a statistics package and computer programming environment and allows anyone to add on or update packages which provide new functions in R.

Benefits of learning R- widely used by most industries and an attractive skill for employers; can handle large complex datasets; provides thousands of packages covering a wide variety of functions including data manipulation, high quality visualisations and statistical modelling; open source software so can run R anywhere at any time; supportive R community to help new users and provide solutions to problems running R packages.

## Getting started with R

Downloading the latest version of R- https://www.r-project.org/

Downloading R from local CRAN mirror https://cran.r-project.org/mirrors.html , New Zealand CRAN- https://cran.stat.auckland.ac.nz/

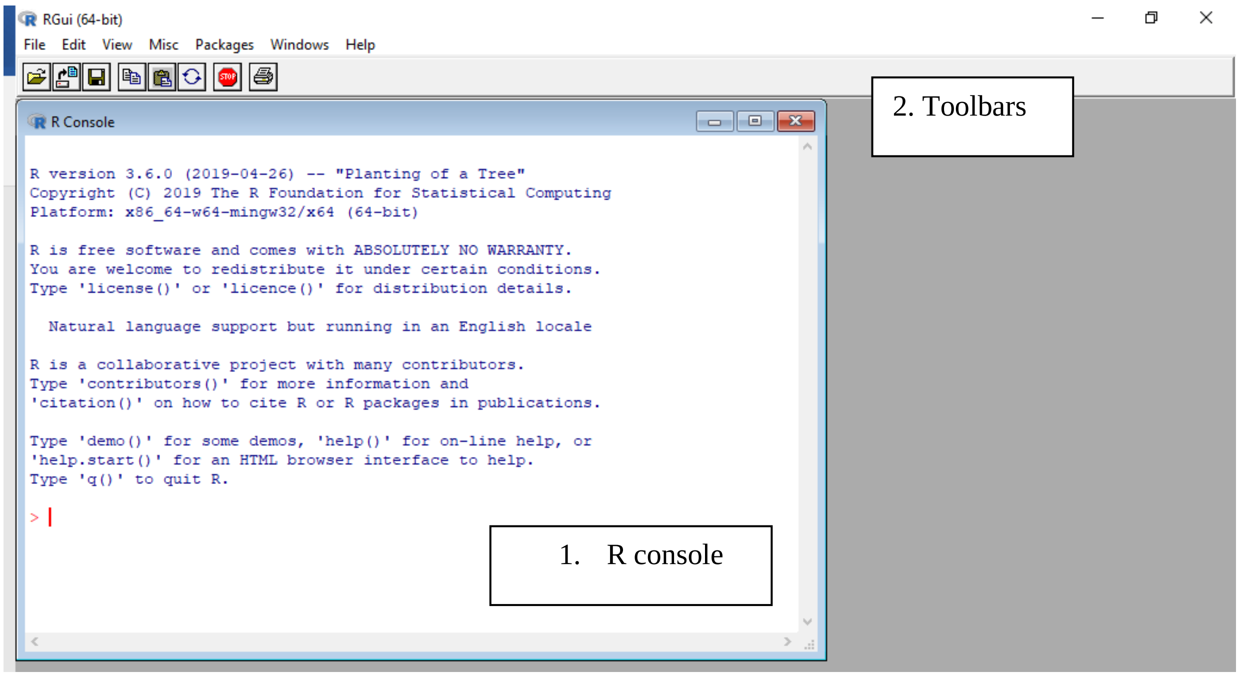
Downloading R studio (desktop) https://rstudio.com/products/rstudio/download/#download

### The R interface

When you first open R you'll be given a window that looks like Figure 1. Two important components to be aware of are the

1. R console. The window that will run all of your code and any results you expect will appear in the R console. When you first open the console it will provide information of what R is and which version you're currently using.

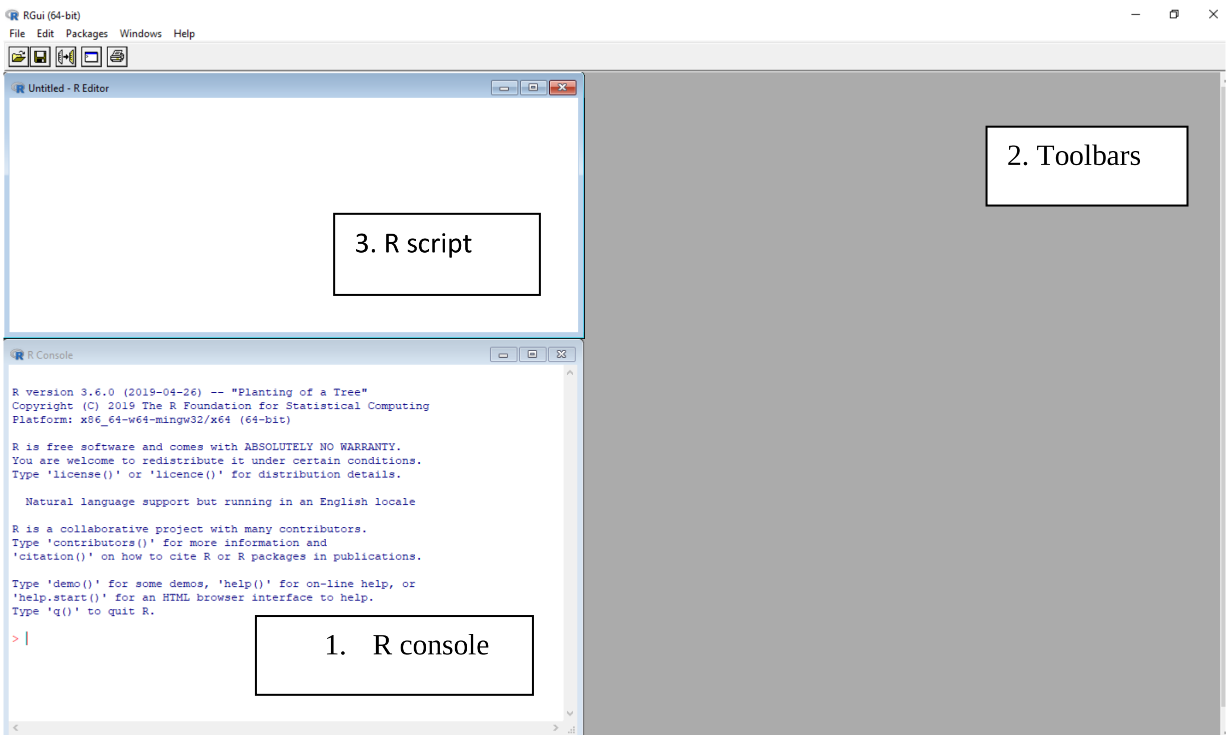
2. Toolbars. The toolbar is important for opening, saving and creating new scripts as well as providing information on what packages you have available and offering a 'Help' list.



```
R version 3.6.0 (2019-04-26) -- "Planting of a Tree"
Copyright (C) 2019 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.

  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.

> 
```

*Opening an R script*

File > New Script



```
R version 3.6.0 (2019-04-26) -- "Planting of a Tree"
Copyright (C) 2019 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.

  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.

> 
```

3. R Script (R editor window). Although we can type R script into the R console, it will not be saved. Any code you wish to save needs to be written inside a script and then

we can save the script for reopening another day to continue working on it. To save an R script go to

File > Save as

And save your scripts somewhere safe and easy to find for when you next need to find them

## Data types and structures

### Types

Everything in R is an ‘object’, but there are 6 basic types of objects or data types in R:

1. Character e.g. “a”, “cat”
2. Numeric e.g. 2, 15.6 (real or decimal)
3. Integer e.g. 2L (the L tells R to store this as an integer)
4. Logical e.g. TRUE, FALSE
5. Complex e.g. 1+4i (complex numbers with real and imaginary parts)

### Structures

R has many data structures. These include:

1. Atomic vectors
2. Lists
3. Matrices
4. Data frames
5. Factors

We won’t go in depth on all of these data types and structures in this workshop, but it’s important to be aware of them as you are likely to use them at some point.

### Vectors

A vector is the most common and basic data structure in R and is pretty much the workhorse of R. Technically, vectors can be one of two types:

1. Atomic vectors
2. Lists

Although the term “vector” most commonly refers to the atomic types not to lists. When we call a vector ‘atomic’, we mean that the vector only holds data of a single data type. Vectors are most commonly of mode character, logical, integer or numeric.

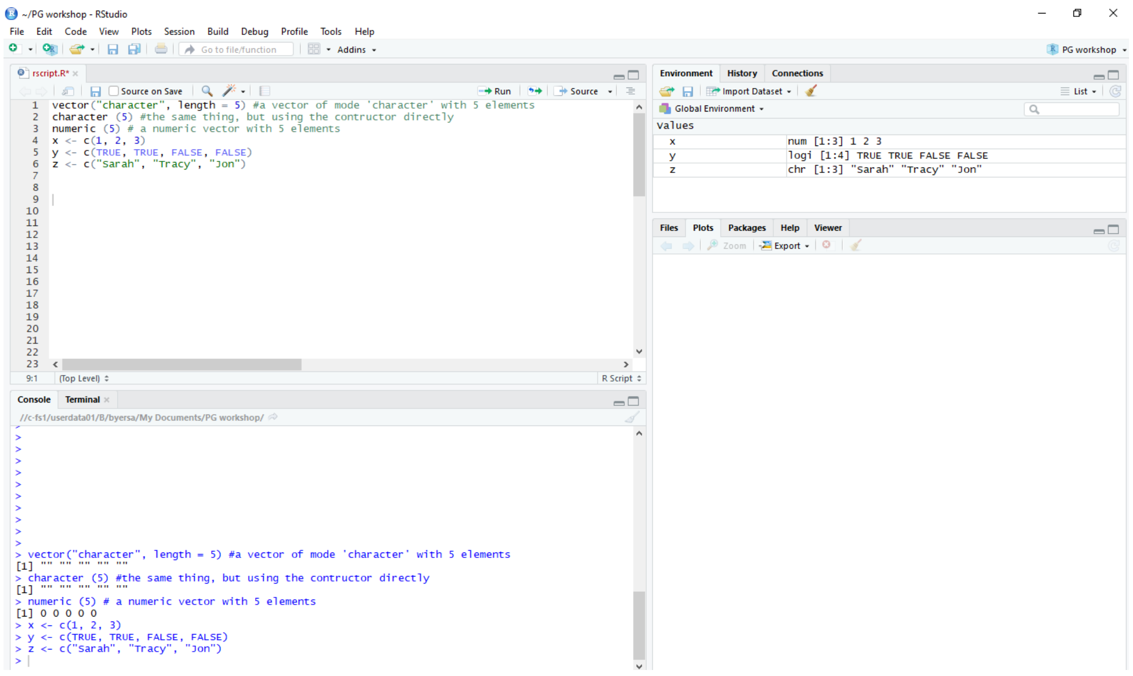
We can create empty vectors using the functions **vector(), character(),** or **numeric()** however it is more common and useful to directly assign values to vectors using the **c()** (for combine) function.

See example below which uses the following code:

```
vector("character", length = 5) # a vector of mode 'character' with
                              5 elements

 character(5) # the same thing, but using the constructor directly
```

```
numeric(5) # a numeric vector with 5 elements

x <- c(1, 2, 3)

y <- c(TRUE, TRUE, FALSE, FALSE)

z <- c("Sarah", "Tracy", "Jon")
```

Side note: This is the first time we've used the # (hashtag) symbol in our code. Anything written after a # (hashtag) will be treated as a comment, and will not be run by R, when you run the line of code. It is extremely useful as it allows you to directly write notes into your code.


~/PG workshop - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
PG workshop
rscript.R*
Source on Save
Run
Source
Environment History Connections
Import Dataset
List
Global Environment
Files Plots Packages Help Viewer
Zoom
Export
(Top Level)
R Script
Console Terminal
//c-fs1/userdata01/B/byersa/My Documents/PG workshop/

```
vector("character", length = 5) #a vector of mode 'character' with 5 elements
character (5) #the same thing, but using the contructor directly
numeric (5) # a numeric vector with 5 elements
x <- c(1, 2, 3)
y <- c(TRUE, TRUE, FALSE, FALSE)
z <- c("Sarah", "Tracy", "Jon")
```

```
Values
x    num [1:3] 1 2 3
y    logi [1:4] TRUE TRUE FALSE FALSE
z    chr [1:3] "Sarah" "Tracy" "Jon"
```

```
>
>
>
>
>
>
>
>
>
>
> vector("character", length = 5) #a vector of mode 'character' with 5 elements
[1] "" "" "" "" ""
> character (5) #the same thing, but using the contructor directly
[1] "" "" "" "" ""
> numeric (5) # a numeric vector with 5 elements
[1] 0 0 0 0 0
> x <- c(1, 2, 3)
> y <- c(TRUE, TRUE, FALSE, FALSE)
> z <- c("Sarah", "Tracy", "Jon")
> 
```

## 1. Environment

We've created three new vectors, and they've appeared in our environment. Notice that the name, type of data (character/numeric/logical), and also the values of the vector are all shown in the environment.

So now we've created some vectors, we can also manipulate them if we wish. Notice that we have re-assigned a new value over x – objects can be written over at any time.

We can use the functions **typeof(), length(), class(),** and **str()** to find out useful information about the vectors (or objects in R in general). We can also use the **c()** (combine) function again if we wish to add some new values to one of the vectors.

```
> typeof(z)
[1] "character"

> str(z)
 chr [1:3] "Sarah" "Tracy" "Jon"

> z <- c(z, "Tom")
> z
[1] "Sarah" "Tracy" "Jon"   "Tom"
```

## Lists

In R lists act as containers. Unlike atomic vectors, the contents of a list are not restricted to a single mode and can encompass any mixture of data types. Lists are sometimes called generic vectors, because the elements of a list can be of any type of R object, even lists containing further lists. This property makes them fundamentally different from atomic vectors. Create lists using **list()** or coerce other objects into lists using **as.list().**

## Matrices

In R matrices are an extension of the numeric or character vectors. They are not a separate type of object but simply an atomic vector with dimensions; that is rows and columns. As with atomic vectors, the elements of a matrix must be of the same data type.

```
m <- matrix(nrow = 2, ncol = 2) #Create a Matrix with 2 rows and 2 columns
```

```
> m
     [,1] [,2]
[1,]   NA   NA
[2,]   NA   NA
```

## Data Frames

Data frames are one of the most important data types in R, and will likely be the one you use the most. These are the de facto data structure for most tabular data and what we most commonly use in statistics.

Essentially a data frame is a special type of list where every element of the list has the same length (i.e. data frame is a “rectangular” list). Imagine an excel spreadsheet with 4 columns and 50 rows, with each cell containing data – this would be a data frame in R.

Data frames can be created by hand using the **data.frame()** function but are more commonly imported into R. You can check the length of a data frame using the **nrow()** (number of rows) or **ncol()** (number of columns) functions.

```
dat <- data.frame(id= letters[1:10], x = 1:10, y = 11:20)

dat
```

```
   id  x  y
1   a  1 11
2   b  2 12
3   c  3 13
4   d  4 14
5   e  5 15
6   f  6 16
7   g  7 17
8   h  8 18
9   i  9 19
10  j 10 20
> nrow(dat)
[1] 10
```

```
> ncol(dat)
[1] 3
```

## Factors

A factor is a vector that can contain only predefined values, and is used to store categorical data. Factors are built on top of integer vectors using two attributes: the class, “factor”, which makes them behave differently from regular integer vectors, and the levels, which defines the set of allowed values. Factors are useful when you know the possible values a variable may take, even if you don’t see all values in a given dataset. Using a factor instead of a character vector makes it obvious when some groups contain no observations:

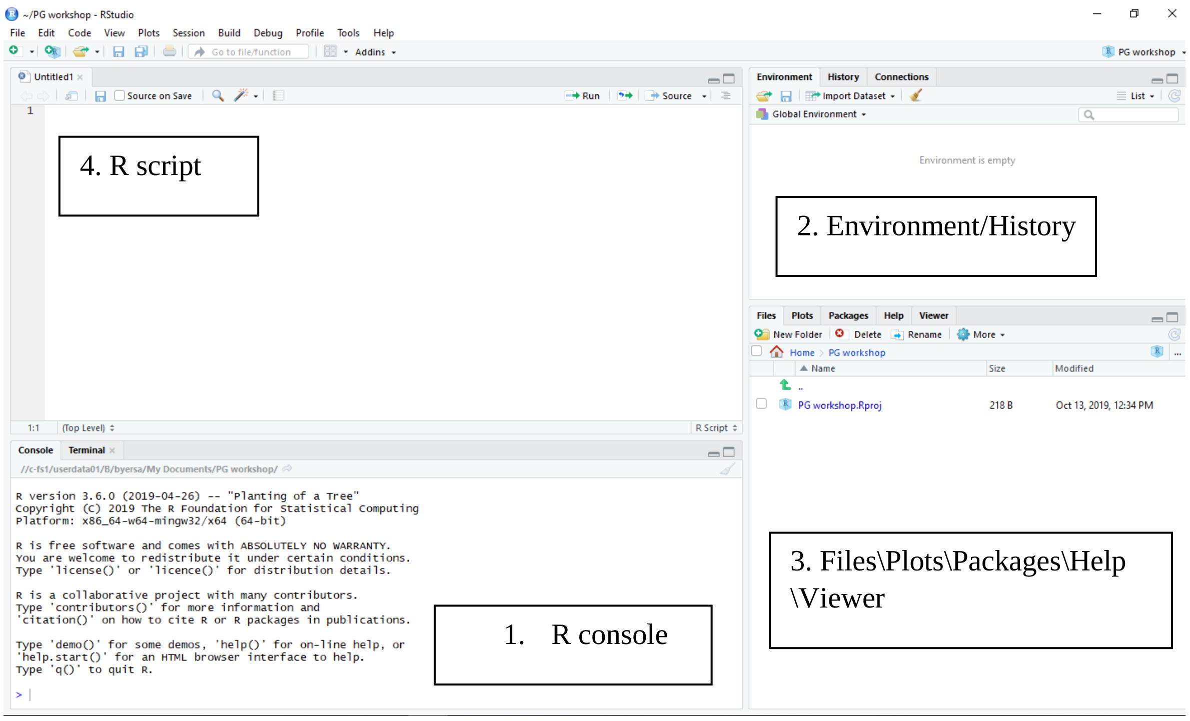
```
sex_char <- c("m", "m", "m")
sex_factor <- factor(sex_char, levels = c("m", "f"))
```

```
> table(sex_char)

sex_char

m

3

> table(sex_factor)

sex_factor

m  f

3  0
```

# Using R studio

R studio provides a new interface for R with additional features which make it easier to use than traditional R. R is the main programme and RStudio uses R to complete its tasks therefore RStudio does not work without R.

RStudio introduces useful features to help you code more smoothly. It uses different text colours identify character strings (green), numbers (blue), comments (green), errors (red), etc. It also introduces standard ‘environment’ and ‘plot’ windows which we’ll explain more below.

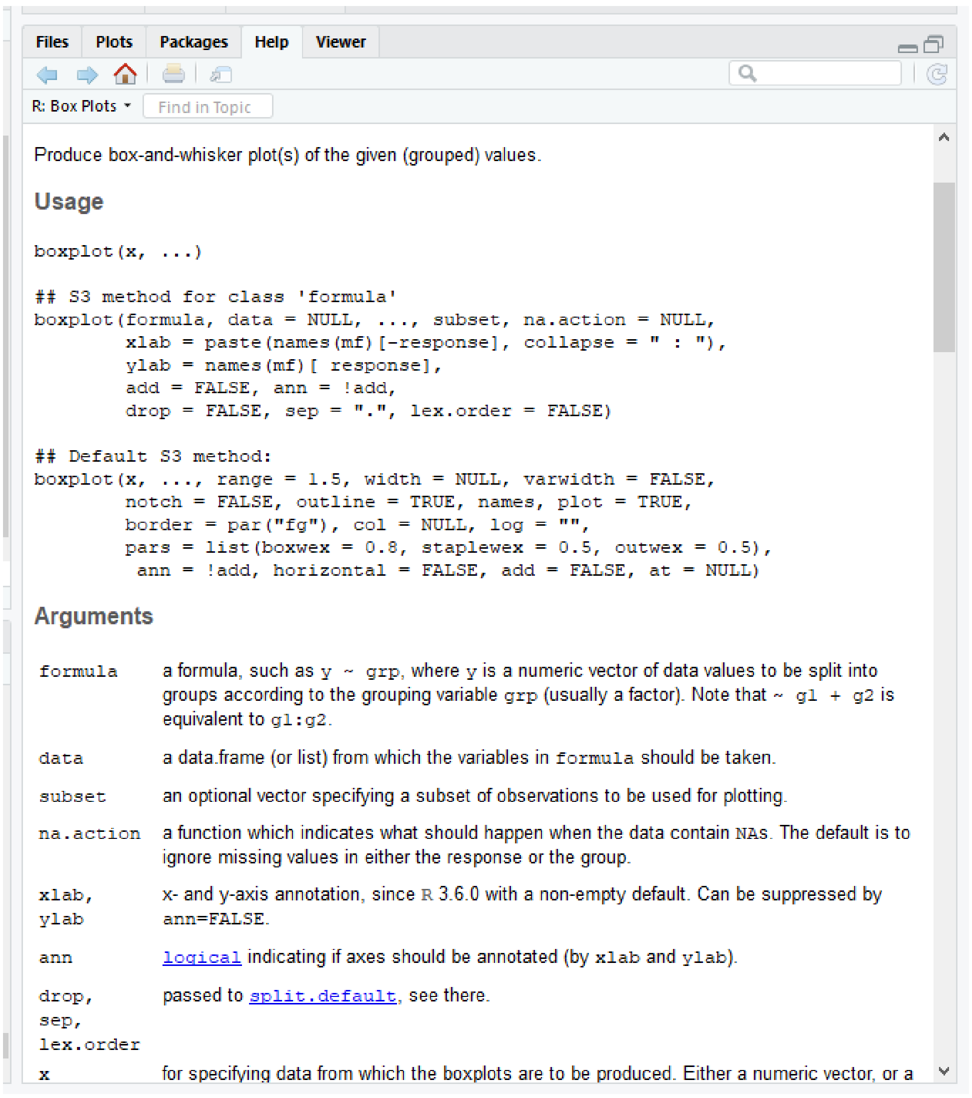

~/PG workshop - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
PG workshop
Untitled1
Source on Save
Run
Source
4. R script
1:1
(Top Level)
R Script
Console
Terminal
//c-fs1/userdata01/B/byersa/My Documents/PG workshop/
1. R console
Environment History Connections
Import Dataset
List
Global Environment
Environment is empty
2. Environment/History
Files Plots Packages Help Viewer
New Folder Delete Rename More
Home › PG workshop
Name Size Modified
PG workshop.Rproj 218 B Oct 13, 2019, 12:34 PM
3. Files\Plots\Packages\Help \Viewer

```
R version 3.6.0 (2019-04-26) -- "Planting of a Tree"
Copyright (C) 2019 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.

> 
```

## Obtaining help for a command

To get help for use of a command, add **?** followed by the command name

```
?boxplot
```



Produce box-and-whisker plot(s) of the given (grouped) values.

**Usage**

```
boxplot(x, ...)

## S3 method for class 'formula'
boxplot(formula, data = NULL, ..., subset, na.action = NULL,
        xlab = paste(names(mf)[-response], collapse = " : "),
        ylab = names(mf)[ response],
        add = FALSE, ann = !add,
        drop = FALSE, sep = ".", lex.order = FALSE)

## Default S3 method:
boxplot(x, ..., range = 1.5, width = NULL, varwidth = FALSE,
        notch = FALSE, outline = TRUE, names, plot = TRUE,
        border = par("fg"), col = NULL, log = "",
        pars = list(boxwex = 0.8, staplewex = 0.5, outwex = 0.5),
         ann = !add, horizontal = FALSE, add = FALSE, at = NULL)
```

**Arguments**

| | |
|---|---|
| `formula` | a formula, such as `y ~ grp`, where `y` is a numeric vector of data values to be split into groups according to the grouping variable `grp` (usually a factor). Note that `~ g1 + g2` is equivalent to `g1:g2`. |
| `data` | a data.frame (or list) from which the variables in `formula` should be taken. |
| `subset` | an optional vector specifying a subset of observations to be used for plotting. |
| `na.action` | a function which indicates what should happen when the data contain `NA`s. The default is to ignore missing values in either the response or the group. |
| `xlab, ylab` | x- and y-axis annotation, since R 3.6.0 with a non-empty default. Can be suppressed by `ann=FALSE`. |
| `ann` | `logical` indicating if axes should be annotated (by `xlab` and `ylab`). |
| `drop, sep, lex.order` | passed to `split.default`, see there. |
| `x` | for specifying data from which the boxplots are to be produced. Either a numeric vector, or a |

# Importing data in R Studio

## Setting up your working directory or folder

RStudio needs to where to look to find and export data and it is best practice to create a folder for any particular analysis to store your R scripts, data files and results. Before you begin you analyses create a folder in your Documents called '**PG R Workshop'**

Now you must tell RStudio that '**PG R Workshop'** is the folder to use for this current R session. On the RStudio main menu, click

*Session > Set working directory > Choose directory*

Navigate to the folder you just created and click on the '**Select folder**' button

## Importing data from Excel

R does not have a built in spreadsheet therefore it is usually easier to enter your data in Excel, export it from Excel in a *.csv* or other text file format and then import it into R.

Import data into RStudio using the **read.csv** command. Type the commnat at the > prompt symbol in the console window. This command needs to know the exact name of your file in quotes.

```
rainfall.dat <- read.csv("rainfall.csv")
```

The <- sign in an assignment sign which assigns the data you have just imported into an R table of data called rainfall.dat. R stores in this data in a workspace which you can see listed if you click on the Environment tab at the top right of you R Studio screen. If you double click on the name you will see the data displayed in a spreadsheet like viewer which you can edit.

## Handling data

### Summarising data

You can use the **summary** command to find the mean (average), median, min, max etc. of you data

```
summary(rainfall.dat)
```

```
    Rainfall       Site 
 Min.   : 59.19   A:12  
 1st Qu.: 61.67   B:12  
 Median : 84.10         
 Mean   : 83.90         
 3rd Qu.:106.29         
 Max.   :107.41         
```

This data frame contains a continuous response variable and one categorical explanatory variable, which is summarised differently.

To display the whole data frame in the console either use the **print** command or enter its name-

```
rainfall.dat
```

```
print(rainfall.dat)
```

For large datasets, use the **head** command to check the first few rows of a data frame-

```
head(rainfall.dat)
```

```
  Rainfall Site
1 105.5372    A
2 106.6686    A
3 106.7742    A
4 103.7552    A
5 106.6457    A
6 107.4057    A
```

To type selected rows or columns, use square brackets **[ ]** and enter the row or column number you wish to display. A number before the comma refers to rows, a number after the comma refers to columns. To display column number 1 of your data frame use-

```
rainfall.dat[ ,1]
```

To display rows only 2 to 7, enter numbers before the comma inside the square brackets-

```
rainfall.dat[2:7, ]
```

The **tapply** command is useful to provide summaries within categories. For example, to find the mean rainfall (column 1) summarised by site (column 2) use-

```
tapply(rainfall.dat[,1], rainfall.dat[,2], mean)
```

```
        A           B
106.08613   61.71708
```

(**mean** can also be substituted with **var** or **sd** to get summarises of variance or standard deviation)

## Plotting data

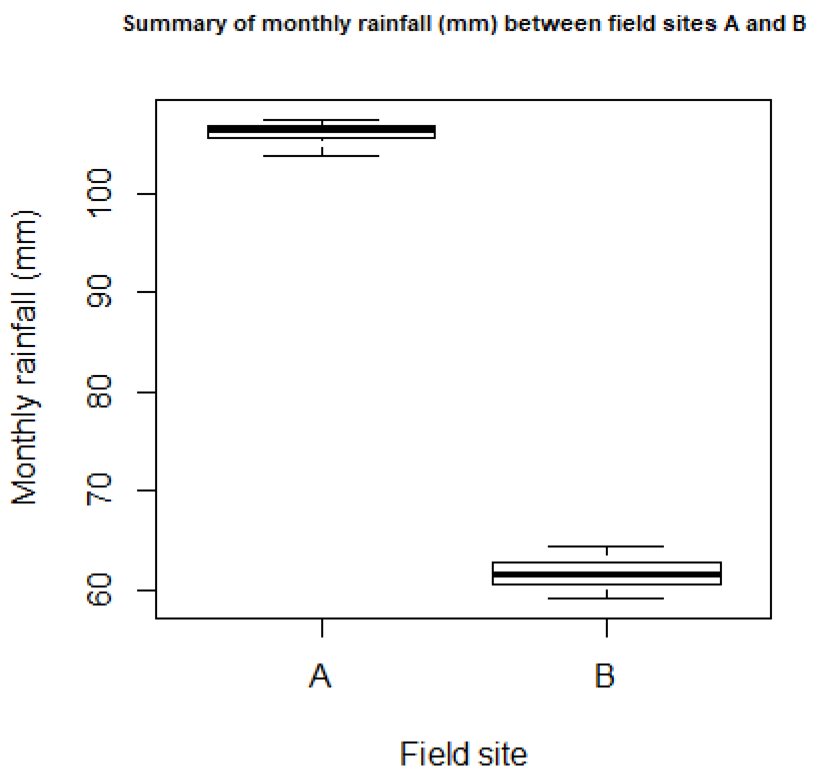
To produce a simple box and whiskers plot showing median and interquartile range to visualise your data, use

```
boxplot(Rainfall ~ Site, data = rainfall.dat)
```

The ~ symbol inside the brackets is to indicate which column is the explanatory on the right side (i.e. site) and which is response on the left (i.e. rainfall).

To add a title (**main**), y axis label (**ylab**) and x axis label (**xlab**), use:

```
boxplot(Rainfall ~ Site, data = rainfall.dat, main="Summary of
monthly rainfall (mm) between field sites A and B", cex.main= 0.7,
ylab = "Monthly rainfall (mm)", xlab = "Field site")
```

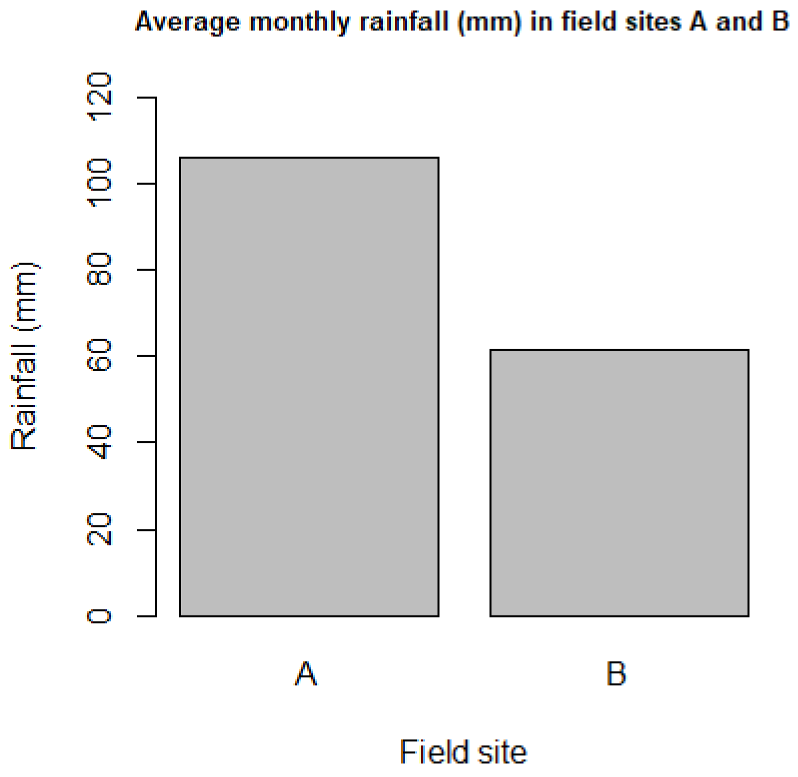


If we wish not to display the variation in rainfall values for each field site, we can simply plot a barplot showing the mean average rainfall. We can do this using the following code

```
data.mean <- tapply(rainfall.dat[,1], rainfall.dat[,2], mean)
barplot(data.mean)
```

To add a title, axis labels, increase the y axis range to 0-120mm and change the title font size we can use

```
barplot(data.mean, main = "Average monthly rainfall (mm) in field
sites A and B", cex.main= 0.8, xlab = "Field site", ylab = "Rainfall
(mm)", ylim = c(0,120))
```



To export this plot, click on the 'Export' button on the graph and either copy it to your clipboard or save as an image or .pdf file.

## Reshaping data

Most univariate analyses in R require the response and explanatory variables to be in different columns. Quite often, data is initially formatted in a way that is not suitable for use in R. The below example shows the plant species diversity along six 10 metre transects across three different meadows-

| MeadowA | MeadowB | MeadowC |
|---|---|---|
| 13 | 28 | 20 |
| 14 | 29 | 26 |
| 13 | 18 | 24 |
| 15 | 33 | 25 |
| 12 | 28 | 27 |

| | | |
|---|---|---|
| 17 | 20 | 24 |

To read this data into your RStudio session, use the **read.csv** command to import the dataset from your working directory-

```
plantdiversity.dat <- read.csv("plantdiversity.csv")
```

```
print(plantdiversity.dat)
```

```
  MeadowA MeadowB MeadowC
1      13      28      20
2      14      29      26
3      13      18      24
4      15      33      25
5      12      28      27
6      17      20      24
```

To analyse this data within R, we must restructure this data so that the response and explanatory variables are contained within two different columns. This can be done using the **stack** command.

```
plantdiversity.stk <- stack(plantdiversity.dat)
```

```
print(plantdiversity.stk)
```

```
   values     ind
1      13 MeadowA
2      14 MeadowA
3      13 MeadowA
4      15 MeadowA
5      12 MeadowA
6      17 MeadowA
7      28 MeadowB
8      29 MeadowB
9      18 MeadowB
10     33 MeadowB
11     28 MeadowB
12     20 MeadowB
13     20 MeadowC
14     26 MeadowC
15     24 MeadowC
16     25 MeadowC
17     27 MeadowC
18     24 MeadowC
```

This has split the response and explanatory data into two columns, however we must now rename the columns into something more clear for our dataset using-

```
colnames(plantdiversity.stk) <- c("plantdiversity", "meadow")
```

View the new column headers and a summary of the dataset use the following **head**, **summary** and **boxplot** commands-

```
head(plantdiversity.stk)
```

```
summary(plantdiversity.stk)
```

```
boxplot(plantdiversity ~ meadow, data = plantdiversity.stk)
```

To obtain the overall plant diversity of the dataset use

```
mean(plantdiversity.stk$plantdiversity)
```

To obtain the **mean**, **median**, variance (**var**) and standard deviation (**sd**) plant diversity for each meadow use

```
tapply(plantdiversity.stk$plantdiversity, plantdiversity.stk$meadow,
mean)
```

```
tapply(plantdiversity.stk$plantdiversity, plantdiversity.stk$meadow,
median)
```

```
tapply(plantdiversity.stk$plantdiversity, plantdiversity.stk$meadow,
var)
```

```
tapply(plantdiversity.stk$plantdiversity, plantdiversity.stk$meadow,
sd)
```

## Summarising continuous data

Here we will look at summarising continuous explanatory data using a data set showing the average monthly crop growth (mm) relative to the amount of monthly rainfall (mm) across 23 different locations in New Zealand.

```
cropgrowth.dat <- read.csv("cropgrowth.csv")
```

```
summary(cropgrowth.dat)
```

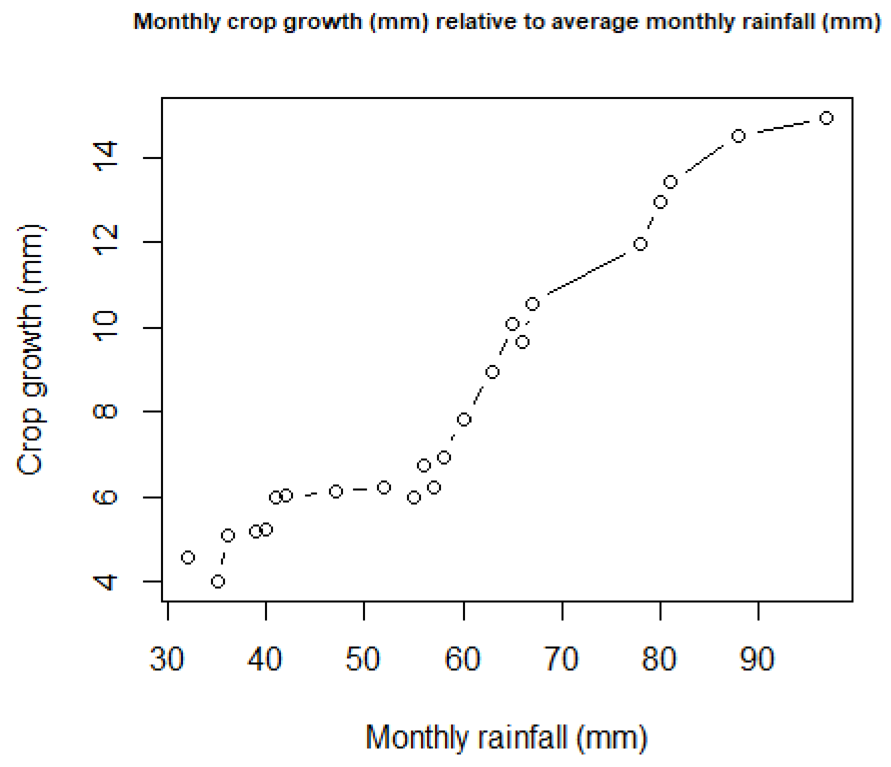
```
     Growth           Rainfall
 Min.   : 3.990   Min.   :32.00
 1st Qu.: 5.975   1st Qu.:41.50
 Median : 6.740   Median :57.00
 Mean   : 8.228   Mean   :58.04
 3rd Qu.:10.340   3rd Qu.:66.50
 Max.   :14.970   Max.   :97.00
```

Unlike the rainfall dataset, R displays the min, mean and max values for your explanatory dataset as it is now a continuous variable.

To visualise the dataset, you can scatterplot it using-

```
plot(Growth ~ Rainfall, data = cropgrowth, main= “Monthly crop
growth (mm) relative to average monthly rainfall (mm)”, cex.main=
0.7, ylab= “Crop growth (mm)”, xlab= “Monthly rainfall (mm)”)
```

In the above code, **main, xlab** and **ylab** are used to add plot title and axis labels; **cex.main** is used to alter title text size and **type=”b”** is used to join data points by lines on plot.



## Exporting data from R studio

### Workspace and history

You can use the **ls()** command to list every set of R objects you have in your workspace environment. For example it can be used to list every object created in this workshop so far-

```
[1] "cropgrowth"          "plantdiversity.dat" "plantdiversity.stk"
 "rainfall.dat"
```

You can also use history () to display a track of what commands you have ran in your analyses so far, which can be helpful for long, complicated analyses

```
history()
```

### Data export

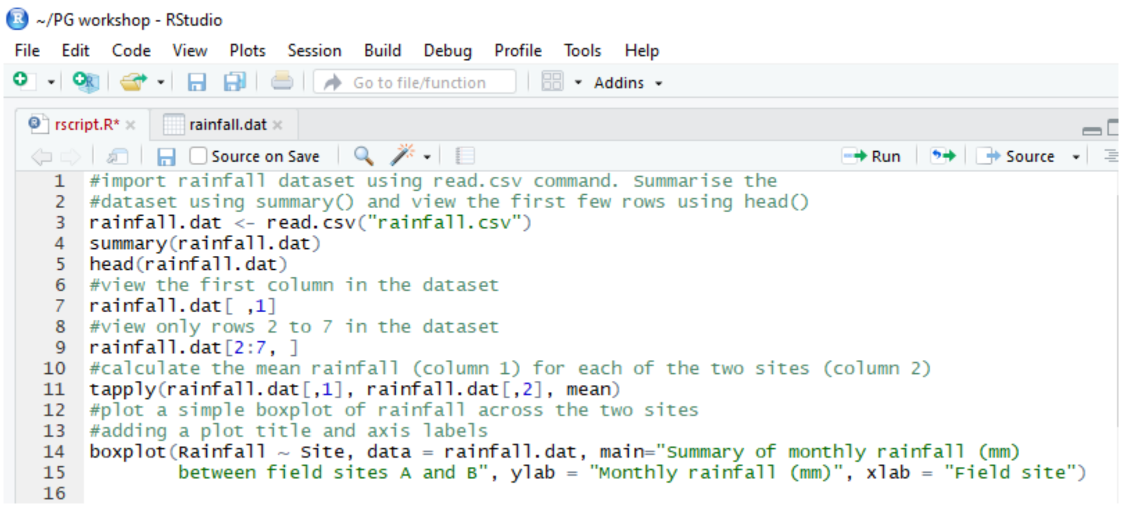
To export your data from R it is best to export it in the .csv file format for use in Excel using the **write.csv** command

```
write.csv(plantdiversity.stk, "plantdiversitystacked.csv")
```

## R scripts

R scripts are plain text files containing a list of R commands which you can annotate with comments to information yourself/others of what you did and why. You can add commnets to R scripts with the # symbol. This is very helpful if you want to repeat analyses and provides a record of what you did. You should develop the habitat of storing all your analyses in R script files.

To open an R script file within RStudio, click File -> Open File and select the appropriate R script file. To run the script, click on the Source button which will run the entire script.



```
#import rainfall dataset using read.csv command. Summarise the
#dataset using summary() and view the first few rows using head()
rainfall.dat <- read.csv("rainfall.csv")
summary(rainfall.dat)
head(rainfall.dat)
#view the first column in the dataset
rainfall.dat[ ,1]
#view only rows 2 to 7 in the dataset
rainfall.dat[2:7, ]
#calculate the mean rainfall (column 1) for each of the two sites (column 2)
tapply(rainfall.dat[,1], rainfall.dat[,2], mean)
#plot a simple boxplot of rainfall across the two sites
#adding a plot title and axis labels
boxplot(Rainfall ~ Site, data = rainfall.dat, main="Summary of monthly rainfall (mm)
         between field sites A and B", ylab = "Monthly rainfall (mm)", xlab = "Field site")
```

## Basic statistics

### t-test

Here we will perform a paired t test to find out if the student scores for Maths are higher than French. We will use the **t.test** function to do this.

First we need to import the scores.csv dataset into our environment

```
scores.dat <- read.csv("scores.csv")
```

Next we can run the t test, using paired = TRUE to indicate it is a paired t test we want to perform. We use the **$** sign to specify which column we want to use from the scores dataset as we are only interested in using one at a time.

```
t.test(scores.dat$Maths, scores.dat$French, paired = TRUE)
```

```
        Paired t-test

data:  scores$Maths and scores$French
t = 1.6922, df = 7, p-value = 0.1344
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.5960154  3.5960154
sample estimates:
mean of the differences
                    1.5
```

### ANOVA via a simple linear model

This section will show how to run basic one way ANOVAs using the rainfall data set previously imported into RStudio as **rainfall.dat**. We will use the **lm()** command to create a linear model and test for significant differences in rainfall across two field sites using analysis of variance (ANOVA).

```
rainfall.lm <- lm(Rainfall ~ Site, data = rainfall.dat)
```

```
anova(rainfall.lm)
```

```
Analysis of Variance Table

Response: Rainfall
          Df  Sum Sq Mean Sq F value    Pr(>F)    
Site       1 11811.7 11811.7  7251.6 < 2.2e-16 ***
Residuals 22    35.8     1.6                      
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Results of this analysis show a significant difference in average monthly rainfall (mm) between the two field sites ($F_{1,22}$= 7251.6; P<0.001).

## ANOVA and multiple comparison tests

We will now use ANOVA on the **plantdiversity.stk** data set to identify if there is a significant difference in plant diversity across the three meadows. Following this we will use the **TukeyHSD** command to run a Tukey multiple comparison test to identify which meadow significantly differs from which.

```
plantdiversity.lm <- lm(plantdiversity ~ meadow, data =
plantdiversity.stk)

anova(plantdiversity.lm)

TukeyHSD(aov(plantdiversity.lm))
```

```
Analysis of Variance Table

Response: plantdiversity
          Df Sum Sq Mean Sq F value    Pr(>F)    
meadow     2 507.11 253.556  17.997 0.0001034 ***
Residuals 15 211.33  14.089                      
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = plantdiversity.lm)

$meadow
                     diff       lwr       upr     p adj
MeadowB-MeadowA 12.000000  6.371038 17.628962 0.0001589
MeadowC-MeadowA 10.333333  4.704371 15.962295 0.0006855
MeadowC-MeadowB -1.666667 -7.295629  3.962295 0.7270279
```

The results of this analysis show a significant difference in plant diversity between meadows ($F_{2,15}$= 17.80, P<0.001). Multiple comparisons tests revealed that plant diversity in Meadow A and B and Meadow A and C were significantly different.

## Simple linear regression

This analysis will use the **cropgrowth.dat** dataset to run a simple linear regression to see if there is a significant correlation between monthly crop growth (mm) and with increased monthly rainfall (mm).

Begin by summarising the plotting the data-

```
summary(cropgrowth.dat)
```

```
plot(Growth ~ Rainfall, data = cropgrowth.dat)
```

To fit the linear regression, use the **lm()** command

```
cropgrowth.lm <- lm(Growth ~ Rainfall, data = cropgrowth.dat)
```

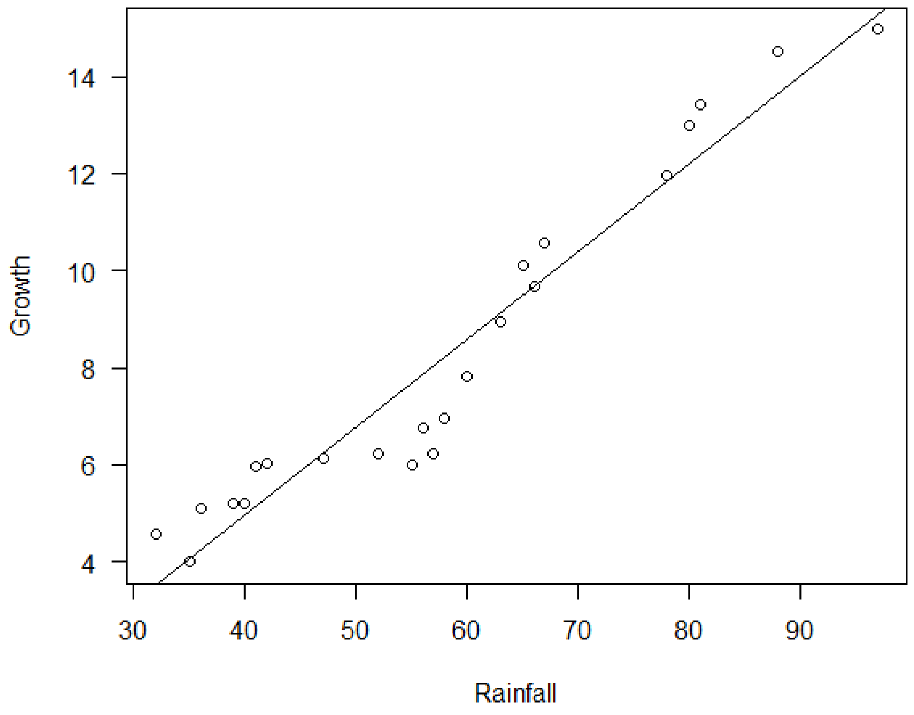
```
summary(cropgrowth.lm)
```

```
Call:
lm(formula = Growth ~ Rainfall, data = cropgrowth)

Residuals:
    Min      1Q  Median      3Q     Max 
-1.8290 -0.5301  0.1204  0.7498  1.0475 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -2.27693    0.65010  -3.502  0.00212 ** 
Rainfall     0.18098    0.01072  16.888 1.07e-13 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.906 on 21 degrees of freedom
Multiple R-squared:  0.9314,	Adjusted R-squared:  0.9281 
F-statistic: 285.2 on 1 and 21 DF,  p-value: 1.072e-13
```

Results of the regression model show the estimated vslue for the gradient is 0.18 and the P-value is <0.001 so the results are highly significant. The overall regression F-statistic is 285.2 and is again highly significant (p<0.001). The adjusted R-squared value is 92.81%, so the model explains over 90% of the variation.

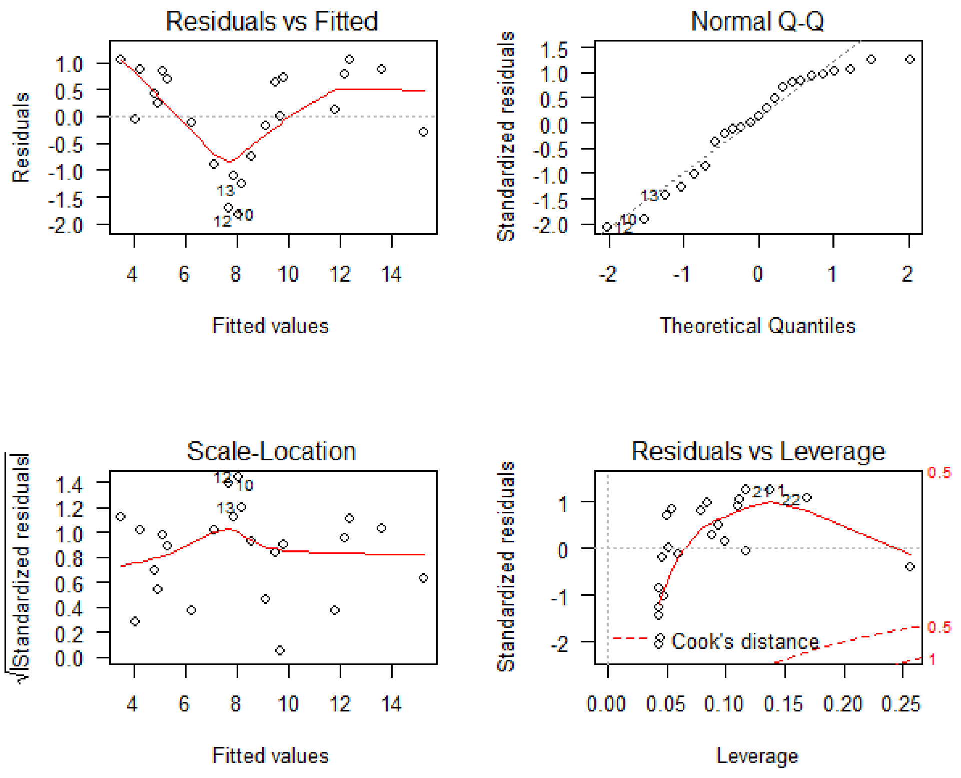
To predict your raw data points with the predicted regression line plot

```
abline(cropgrowth.lm)
```



We can use the following command to view the model diagnostic plots to check the model is a good fit for the data use. The **par(mfrow=c(2,2)** command simply adds the 4 separate diagnostic plots into one plot.

```
par(mfrow=c(2,2))
plot(cropgrowth.lm)
```



## Useful R functions and packages

| GENERAL FUNCTIONS | DESCRIPTION |
|---|---|
| builtins() | List all built-in functions |
| options() | Set options to control how R computes & displays results |
| ?NA | Help page on handling of missing data values<br>abs(x) The absolute value of "x" |
| append() | Add elements to a vector |
| c(x) | A generic function which combines its arguments |
| cat(x) | Prints the arguments |
| cbind() | Combine vectors by row/column (cf. "paste" in Unix) |
| diff(x) | Returns suitably lagged and iterated differences |
| gl() | Generate factors with the pattern of their levels |
| grep() | Pattern matching |
| identical() | Test if 2 objects are *exactly* equal |
| jitter() | Add a small amount of noise to a numeric vector |
| julian() | Return Julian date |
| length(x) | Return no. of elements in vector x |
| ls() | List objects in current environment |
| mat.or.vec() | Create a matrix or vector<br>paste(x) Concatenate vectors after converting to character |
| range(x) | Returns the minimum and maximum of x |
| rep(1,5) | Repeat the number 1 five times |
| rev(x) | List the elements of "x" in reverse order |
| seq(1,10,0.4) | Generate a sequence (1 -> 10, spaced by 0.4) |
| sequence() | Create a vector of sequences |
| sign(x) | Returns the signs of the elements of x |
| sort(x) | Sort the vector x |
| order(x) | List sorted element numbers of x |
| tolower(),toupper() | Convert string to lower/upper case letters |
| unique(x) | Remove duplicate entries from vector<br>system("cmd") Execute "cmd" in operating system (outside of R) |

| | |
|---|---|
| vector() | Produces a vector of given length and mode |
| formatC(x) | Format x using 'C' style formatting specifications |
| floor(x), ceiling(x), round(x), signif(x), trunc(x) | Rounding functions |
| Sys.getenv(x) | Get the value of the environment variable "x" |
| MATHMATECAL FUNCTIONS | DESCRIPTION |
| Sys.putenv(x) | Set the value of the environment variable "x" |
| Sys.time() | Return system time |
| Sys.Date() | Return system date |
| getwd() | Return working directory |
| setwd() | Set working directory |
| ?files | Help on low-level interface to file system |
| list.files() | List files in a give directory |
| file.info() | Get information about files |
| pi,letters,LETTERS | Pi, lower & uppercase letters, e.g. letters[7] = "g" |
| month.abb,month.name | Abbreviated & full names for months Maths |
| log(x),logb(),log10(),log2(),exp(),expm1(),log1p() ,sqrt() | Transformations: Log, Exponentials, Square Root |
| cos(),sin(),tan(),acos(),asin(),atan(),atan2() | Trigonometry |
| cosh(),sinh(),tanh(),acosh(),asinh(),atanh() | Hyperbolic functions |
| union(),intersect(),setdiff(),setequal() | Set operations |
| +,-,*,/,^,%%,%/% | Arithmetic operators |
| ?Special | Help on special functions related to beta and gamma functions |
| ?Syntax | Help on R syntax and giving the precedence of operators |
| ?regex | Help on regular expressions used in R |
| ?Paren | Help on parentheses |
| ?Mod | Help on functions which support complex arithmetic in R |
| ?Logic | Help on logical operators |
| ?Extract | Help on operators acting to extract or replace subsets of vectors |
| ?Control | Help on control flow statements (e.g. if, for, while) |
| sum() | Sum or total (add things together) |
| integrate() | Adaptive quadrature over a finite or infinite interval. |
| deriv() | Symbolic and algorithmic derivatives of simple expressions |
| eigen() | Computes eigenvalues and eigenvectors |
| <,>,<=,>=,==,!= | Comparison operators |

| GRAPHICAL FUNCTIONS | |
|---|---|
| help(package=graphics) | List all graphics functions |
| plot() | Generic function for plotting of R objects |
| par() | Set or query graphical parameters |
| curve(5*x^3,add=T) | Plot an equation as a curve |
| points(x,y) | Add another set of points to an existing graph |
| arrows() | Draw arrows [see errorbar script] |
| abline() | Adds a straight line to an existing graph |
| lines() | Join specified points with line segments |
| segments() | Draw line segments between pairs of points |
| hist(x) | Plot a histogram of x |
| pairs() | Plot matrix of scatter plots |
| matplot() | Plot columns of matrices |
| ?device | Help page on available graphical devices |
| postscript() | Plot to postscript file |
| pdf() | Plot to pdf file |
| png() | Plot to PNG file |
| jpeg() | Plot to JPEG file |
| X11() | Plot to X window |
| persp() | Draws perspective plot |
| contour() | Contour plot |
| image() | Plot an image |
| STATISTICAL FUNCTIONS | DESCRIPTION |
| lm | Fit linear model |
| glm | Fit generalised linear model |
| nls | Non-linear (weighted) least-squares fitting |
| lqs | "library(MASS)" resistant regression |
| aov() | Analysis of Variance (ANOVA) |
| optim | General-purpose optimisation |
| optimize | 1-dimensional optimisation |
| constrOptim | Constrained optimisation |
| nlm | Non-linear minimisation |
| nlminb | More robust (non-)constrained non-linear minimisation |
| help(package=stats) | List all stats functions |
| ?Chisquare | Help on chi-squared distribution functions |
| ?Poisson | Help on Poisson distribution functions help(package=survival) Survival analysis |
| cor.test() | Perform correlation test |

| | |
|---|---|
| cumsum(); cumprod(); cummin(); cummax() | Cumulative functions for vectors |
| density(x) | Compute kernel density estimates |
| ks.test() | Performs one or two sample Kolmogorov-Smirnov tests |
| loess(), lowess() | Scatter plot smoothing |
| mad() | Calculate median absolute deviation |
| mean(x), weighted.mean(x), median(x), min(x), max(x), quantile(x) | Summary Statistics |
| rnorm(), runif() | Generate random data with Gaussian/uniform distribution |
| splinefun() | Perform spline interpolation |
| smooth.spline() | Fits a cubic smoothing spline |
| sd() | Calculate standard deviation |
| summary(x) | Returns a summary of x: mean, min, max etc. |
| t.test() | Student's t-test |
| var() | Calculate variance |
| sample() | Random samples & permutations |
| ecdf() | Empirical Cumulative Distribution Function |
| qqplot() | Quantile-quantile plot |

## Community resources

There are heaps of resources online to help out with learning to code, or just fixing problems you encounter when coding – here are some of them!

Learning to code from scratch

https://www.datacamp.com/ - This website provides free interactive tuition for a beginner R course. You have to pay if you wish to continue learning after the beginner material though.

Fixing Problems

https://www.google.com/ - First a foremost google it! If you're having a problem in R then it's likely someone else has encountered and solved the same problem already. Either copy and paste an error message into google, or try and describe your problem and you're likely to find a solution!

https://stackoverflow.com/ and https://stackexchange.com/ - These are very common websites to search for answers to questions you may have. They will likely pop up in google searches, but if you're really stuck and cannot find the answer these websites allow you to directly ask questions and have them answered by a very active coding community.

And finally, drop in to the library! Drop in sessions are every day from 10.30am-11.30am, and you are welcome to book appointments with us outside of these times using https://ltl.lincoln.ac.nz/advice/study-skills/book-a-workshop-or-appointment/

## Full code used today

This is also saved as an R script in PG workshop folder

```
#Data types and structures

# a vector of mode 'character' with 5 elements

vector("character", length = 5)

# the same thing, but using the constructor directly

character(5)

# a numeric vector with 5 elements

numeric(5)

x <- c(1, 2, 3)

y <- c(TRUE, TRUE, FALSE, FALSE)

z <- c("Sarah", "Tracy", "Jon")

#finding out information about vectors

typeof(z)

str(z)

#adding new values to a vector

z <- c(z, "Tom")

#Create a Matrix with 2 rows and 2 columns

m <- matrix(nrow = 2, ncol = 2)

#Create a data frame with 3 columns of one categorical and two
numeric variables

dat <- data.frame(id= letters[1:10], x = 1:10, y = 11:20)

dat

#Setting factors for vectors with predefined values and categorical
variables.

#Shown here as the values are either male or female

sex_char <- c("m", "m", "m")

sex_factor <- factor(sex_char, levels = c("m", "f"))

#Help function

?boxplot

#Importing and viewing data

rainfall.dat <- read.csv("rainfall.csv")
```

```
summary(rainfall.dat)

rainfall.dat

print(rainfall.dat)

#Selecting column/rows to view

rainfall.dat[ ,1]

rainfall.dat[2:7, ]

#calculating data summaries, here calculating the mean

tapply(rainfall.dat[,1], rainfall.dat[,2], mean)

#Plotting data summaries

boxplot(Rainfall ~ Site, data = rainfall.dat)

boxplot(Rainfall ~ Site, data = rainfall.dat, main="Summary of
monthly rainfall (mm) between field sites A and B", cex.main = 0.7,
ylab = "Monthly rainfall (mm)", xlab = "Field site")

data.mean <- tapply(rainfall.dat[,1], rainfall.dat[,2], mean)

barplot(data.mean)

barplot(data.mean, main = "Average monthly rainfall (mm) in field
sites A and B", cex.main= 0.8, xlab = "Field site", ylab = "Rainfall
(mm)", ylim = c(0,120))

#reshaping data

plantdiversity.dat <- read.csv("plantdiversity.csv")

print(plantdiversity.dat)

plantdiversity.stk <- stack(plantdiversity.dat)

print(plantdiversity.stk)

#changing column names

colnames(plantdiversity.stk) <- c("plantdiversity", "meadow")

head(plantdiversity.stk)

summary(plantdiversity.stk)

boxplot(plantdiversity ~ meadow, data = plantdiversity.stk)

#plotting mean of overall dataset

mean(plantdiversity.stk$plantdiversity)

#plotting data summaries by each categorical variable

tapply(plantdiversity.stk$plantdiversity, plantdiversity.stk$meadow,
mean)

tapply(plantdiversity.stk$plantdiversity, plantdiversity.stk$meadow,
median)
```

```
tapply(plantdiversity.stk$plantdiversity, plantdiversity.stk$meadow,
var)

tapply(plantdiversity.stk$plantdiversity, plantdiversity.stk$meadow,
sd)

#summarising continuous data

cropgrowth.dat <- read.csv("cropgrowth.csv")

summary(cropgrowth.dat)

plot(Growth ~ Rainfall, data = cropgrowth.dat, main= "Monthly crop
growth (mm) relative to average monthly rainfall (mm)", cex.main=
0.7, ylab= "Crop growth (mm)", xlab= "Monthly rainfall (mm)")

#viewing workspace history

ls()

history()

#data export

write.csv(plantdiversity.stk, "plantdiversitystacked.csv")

#paired t tests

scores.dat <- read.csv("scores.csv")

t.test(scores.dat$Maths, scores.dat$French, paired = TRUE)

#anova via linear model

rainfall.lm <- lm(Rainfall ~ Site, data = rainfall.dat)

anova(rainfall.lm)

#anova and multiple comparisons

plantdiversity.lm <- lm(plantdiversity ~ meadow, data =
plantdiversity.stk)

anova(plantdiversity.lm)

TukeyHSD(aov(plantdiversity.lm))

#simple linear regression

summary(cropgrowth.dat)

plot(Growth ~ Rainfall, data = cropgrowth.dat)

cropgrowth.lm <- lm(Growth ~ Rainfall, data = cropgrowth.dat)

summary(cropgrowth.lm)

abline(cropgrowth.lm) #add regression line to plot

par(mfrow=c(2,2)) #combined 4 diagnostic plots into one plot image

plot(cropgrowth.lm) #plot diagnostic plots
```

October 2019