Programming & Data Analysis with ‘R’

Diarmuid O'Briain

Document Outline

This document brings together a collection of different *R* experiences. It documents work carried out during a week long training on "*Statistical Data Analysis using R-software*" that took place from 24 - 28 September 2018 at the Directorate of Research and Graduate Training (DRGT), Makerere University, Kampala, Uganda that was delivered by Associate Professor Matthew Low and Dr. Matthew Hiron of the Swedish University of Agricultural Sciences (SLU) to develop the programming aspect of *R* with a focus on the analysis of quantitative data sets. I have included a section on qualitative data analysis as *R* has some tools in that area too.

The document explores the fundamentals of how to use *R*. Installing *R*, the structure of *R*, vectors, matrices, arrays, lists, tables, data-frames as well as flow control and user-defined functions. Working with quantitative data, linear models, predictions, probability distributions, distribution models and generalised linear mixed models. Plotting data for visual output. Finally both inductive and deductive approaches to the analysis of qualitative data is explored.

Diarmuid Ó Briain

Table of Contents

Illustration Index

Table of Abbreviations

AIC	Akaike Information Criterion
ANCOVA	Analysis of covariance
ANOVA	Analysis of Variance
CO_2	Oxygen
CRAN	Comprehensive R Archive Network
CSS	Cascading Style Sheets
CSV	Comma Separated file
DRGT	Directorate of Research and Graduate Training
DV	Dependent Variable
FUN	Function
GLM	Generalised Linear Model
GLMM	Generalised Linear Mixed Model
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IV	Independent Variable
LMM	Linear Mixed-effects Model
MANOVA	Multivarite ANOVA
NLME	Non-linear Mixed-Effects Model
p-test	Statistical method used to assess one or more hypotheses within a population or a proportion within a population
R	Open-source free statistical programming language used by scientists
REML	Restricted (or Residual, or Reduced) Maximum Likelihood
RQDA	R Qualitative Data Analysis
SD	Standard Deviation
SE	Standard Errors
SLU	Swedish University of Agricultural Sciences
SQL	Structured Query Language
Student t-test	t-test introduced by William Sealy Gosset
SVG	Scalable Vector Graphics
t-test	Analysis of two populations means through the use of statistical examination
Tukey	Tukey's Honest Significant Difference
Welch t-test	Adaptation of Student's t-test
YAML	YAML Ain't Markup Language

1. Installing R

This document is based on implementation on the GNU/Linux operating system. It assumes a Debian GNU/Linux based distribution like Debian, Ubuntu, Linux Mint, Elementary OS or a host of others. If the GNU/Linux operating system is RedHat or Fedora based like CentOS then the *yum* package manager is used. Replace the *sudo apt-get install <package name>* with *sudo yum install <package name>*. For other operating systems like Apple macOS or https://cloud.r-project.org/bin/windows/ Microsoft please consult the relevant r-project links.

R can be used directly from the terminal shell and text editor tools like *xed* and *kate* have built in highlight modes, typically under the *Scientific* sections of these programs. Alternatively install an Integrated Development Environment (IDE) like *Rstudio*.

To operate with the terminal shell and text editors install the following base package.

```
$ sudo apt-get install r-base
```

If *Rstudio* is required the also install the following packages.

```
$ sudo apt-get install rstudio
$ sudo apt-get install r-cran-rstudioapi
```

This document will proceed with the assumption of a terminal shell and a text editor.

2. Introduction

2.1 Simple R script

Starting out with a simple *R* script and redirect it into the *R* interpreter. Copy from *cat* to *EOM* on its own and paste to a terminal shell, best to do this in a directory specifically for the purpose. This creates a file called *HelloWorld.R* in the working directory. Redirect the file to the R interpreter.

```
$ cat << EOM >> HelloWorld.R
## run this code in R
## Hello World script
plot(x=1, y=1, typ="n", xlab="", ylab="", bty="n", xaxt="n", yaxt="n", ylim=c(-1,2))
text(x=1, y=1,"'R' is a language for statistical analysis", cex=2, col="red")
text(x=1, y=0.5, "R: Hello World", cex=3, col="blue")
EOM

$ R < test.R
```

A file called *Rplots.pdf* is generated in the working directory.

Illustration 1: R: Hello World

2.2 Another simple script

Some basic maths in another script.

```
$ cat << EOM >> Introduction.R
# R introduction
1+3
4*7
a=3
a+7
EOM

$ R < Introduction.R --vanilla

R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (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.

> # R introduction
> 1+3
[1] 4

> 4*7
[1] 28

> a=3
> a+7
[1] 10
```

2.3 Start a new script

In both of these cases the script was redirected to the R interpreter. It is also possible to make the script executable and allow it find the interpreter by the inclusion of a shebang (#!) line on the first line referring the operating system to the to the path of the *R* interpreter. Make the file executable and run it.

```
$ cat << EOM >> vi my_first_script.R
#!/usr/bin/Rscript
# R introduction

1 + 3
4 * 7
a = 3
a + 7
EOM

$ chmod +x my_first_script.R
$ ./my_first_script.R
[1] 4
[1] 28
[1] 10
```

2.4 Running an R shell

An R shell can be ran to execute a small number of commands. Many of the examples in this document are executed in this way.

```
$ R

R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (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.

>
```

To run *R* without the message each time add the *--quiet* option.

```
$ R --quiet
>
```

2.5 R errors

Consider the example below. As *b* has not been defined *R* has no way of carrying out the instruction and will therefore throw back an error as demonstrated.

```
$ R --quiet
> b + 7
Error: object 'b' not found
```

2.6 Commenting

Annotate code as you write, it makes it easier to read later. The annotations are preceded with a **#** symbol. The *Rscript* interpreter will ignore the annotations.

```
$ cat << EOM >> vi my_first_annotate.R
#!/usr/bin/Rscript
a = list(c(1,2,3,4),      # A vector
         "Some",          # Add characters
         "Words",         # Add a second set of characters
         c(5,6,7,8)       # A second vector
        )

## Exercise
str (a)                   # Output the structure of 'a'
m = lapply(a[1], mean)    # Apply the mean function to 1st list in 'a'
n = lapply(a[4], max)     # Apply the max function to 4st list in 'a'
a$means = c(m,n)          # Add a new item to the list
print(a)
EOM

$ chmod +x my_first_annotate.R
```

```
$ ./my_first_annotate.R
List of 4
 $ : num [1:4] 1 2 3 4
 $ : chr "Some"
 $ : chr "Words"
 $ : num [1:4] 5 6 7 8
[[1]]
[1] 1 2 3 4

[[2]]
[1] "Some"

[[3]]
[1] "Words"

[[4]]
[1] 5 6 7 8

$means
$means[[1]]
[1] 2.5

$means[[2]]
[1] 8
```

2.6.1 Multiline commenting

Many programming languages offer the ability to do multiline commenting. This is useful if there are some lines in the script that you want the script to skip without the need to put *#* before each line in the block. In this case use the structure below around the block. In this case the test is always FALSE and therefore the lines are skipped.

```
if (FALSE){

Lines to be ignored.

}
```

To demonstrate. Watch what happens if this structure is added to the script. The lines between 'if(FALSE){' and '}' are bypassed by the *R* interpreter.

```
$ cat << EOM >> multiline_comment.R
#!/usr/bin/Rscript

a = list(c(1,2,3,4),      # A vector
         "Some",          # Add characters
         "Words",         # Add a second set of characters
         c(5,6,7,8)       # A second vector
        )

## Exercise
str (a)                   # Output the structure of 'a'
if(FALSE){
m = lapply(a[1], mean)    # Apply the mean function to 1st list in 'a'
n = lapply(a[4], max)     # Apply the max function to 4st list in 'a'
a$means = c(m,n)          # Add a new item to the list
}
print(a)
EOM
```

```
$ R < my_first_annotate.R
List of 4
 $ : num [1:4] 1 2 3 4
 $ : chr "Some"
 $ : chr "Words"
 $ : num [1:4] 5 6 7 8
[[1]]
[1] 1 2 3 4

[[2]]
[1] "Some"

[[3]]
[1] "Words"

[[4]]
[1] 5 6 7 8
```

So what happened?

2.7 Listing the existing objects in R

ls(): Returns a vector of character strings giving the names of the objects in the specified environment.

objects(): Same as *ls()*.

```
> ls()
 [1] "df"        "model1"    "model2"    "model3"    "model4"    "realmodel"
 [7] "v"         "v1"        "v2"        "v3"        "v4"        "x"
```

2.7.1 Clearing the existing objects in R

rm(): Removes objects specified successively as character strings, or in a character vector *list*, or through a combination of both.

remove(): Same as *rm()*.

To clear the environment of objects, supply a *list* of objects to the *rm()* or *remove()* functions.

```
> ls()
 [1] "df"        "model1"    "model2"    "model3"    "model4"    "realmodel"
 [7] "v"         "v1"        "v2"        "v3"        "v4"        "x"

> rm(list=ls())

> ls()
character(0)
```

or alternatively:

```
> objects()
 [1] "df"        "model1"    "model2"    "model3"    "model4"    "realmodel"
 [7] "v"         "v1"        "v2"        "v3"        "v4"        "x"

> remove(list=objects())

> objects()
character(0)
```

3. R structure

rnorm(): random generation for the normal distribution.

rnorm(5,mean=0,sd=1)

Note: *rnorm(n, mean = , sd =)* is used to generate n normal random numbers with arguments mean and standard deviation.

```
> rnorm(5,mean=0,sd=1)
[1]  1.3136807 -1.2612950  0.9052489  0.8711972 -1.7449344

> plot(rnorm(5,mean=0,sd=1))
```

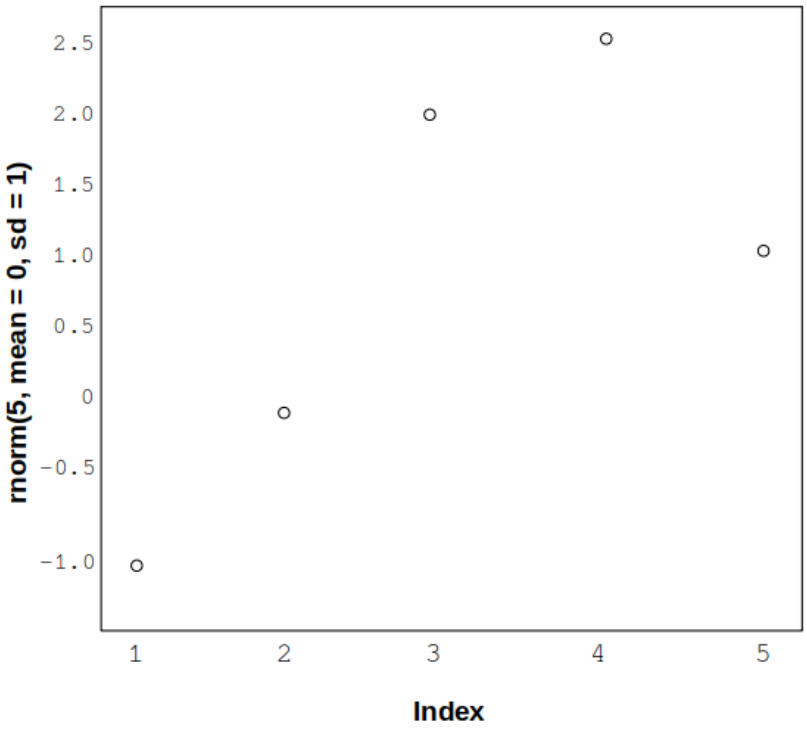


Illustration 2: rnorm()

function(*function(object*))

Note: The *R abs()* method is one of the R mathematics functions, which is used to return the absolute positive value of an individual number, or an expression.

```
> abs(rnorm(5,mean=0,sd=1))
[1] 0.07609716 0.57743538 1.10299258 0.42479172 1.41277626
```

function(*function(function(object*)))

```
> mean(abs(rnorm(100,mean=0,sd=1)))
[1] 0.8621388
```

4. Getting help with functions

There are a number of helpful tools in *R* to get more information.

- **args():** - Lists the arguments of a function give as an argument.
- **help(), ?:** - Get the man page for the function given as an argument.
- **example():** - Get examples of how the function given as an argument can be used.

4.1 Arguments of a function

To understand a new function and one needs the available arguments within the function.

function(*arguments*)

```
> args(rnorm)
function (n, mean = 0, sd = 1)
NULL
```

- **n** - User specified, i.e. number of numbers
- **mean** - By default the mean = 0
- **sd** - Default standard deviation = 1

4.2 help()

```
> help() or
persp                  package:graphics                  R Documentation

Perspective Plots

Description:

     This function draws perspective plots of a surface over the x-y
     plane.  'persp' is a generic function.

Usage:

     persp(x, ...)
```

4.3 example()

Example function gives examples of the function given to it as an argument.

```
> example(mean)

mean> x <- c(0:10, 50)

mean> xm <- mean(x)

mean> c(xm, mean(x, trim = 0.10))
[1] 8.75 5.50
```

More information can be found online at:

quick-r

inside-r

r-graph gallery

5. R packages

5.1 The Comprehensive R Archive Network (CRAN)

The CRAN is a resource of packages that can be added to *R*.

The Comprehensive R Archive Network website

5.2 Installing Packages in R

To install packages an internet connection is necessary.

```
> install.packages("lme4")
```

Once the package is downloaded it is there but the package must be *loaded* to use it.

```
> library(lme4)
```

In a script that has a dependency on a package it is helpful to have a line at the top of the file that causes the package to be installed if the package is not already installed. In the example, if the package *lme4* is not installed then *R* will install it first before continuing with the script. If it is installed then the line is ignored. The *library('lme4')* line loads the library such that the functions within the package *lme4* become available within the program.

```
> if(!require(lme4)){install.packages("lme4")}

> library(lme4)
```

5.3 Check installed packages

Check packages already installed with the *ip* command.

```
> ip
           Package      LibPath              Version   Priority
base       "base"       "/usr/lib/R/library" "3.4.4"   "base"
boot       "boot"       "/usr/lib/R/library" "1.3-20"  "recommended"
class      "class"      "/usr/lib/R/library" "7.3-14"  "recommended"
cluster    "cluster"    "/usr/lib/R/library" "2.0.6"   "recommended"
codetools  "codetools"  "/usr/lib/R/library" "0.2-15"  "recommended"
compiler   "compiler"   "/usr/lib/R/library" "3.4.4"   "base"
datasets   "datasets"   "/usr/lib/R/library" "3.4.4"   "base"
foreign    "foreign"    "/usr/lib/R/library" "0.8-70"  "recommended"
graphics   "graphics"   "/usr/lib/R/library" "3.4.4"   "base"
grDevices  "grDevices"  "/usr/lib/R/library" "3.4.4"   "base"
grid       "grid"       "/usr/lib/R/library" "3.4.4"   "base"
KernSmooth "KernSmooth" "/usr/lib/R/library" "2.23-15" "recommended"
lattice    "lattice"    "/usr/lib/R/library" "0.20-35" "recommended"
MASS       "MASS"       "/usr/lib/R/library" "7.3-49"  "recommended"
Matrix     "Matrix"     "/usr/lib/R/library" "1.2-12"  "recommended"
methods    "methods"    "/usr/lib/R/library" "3.4.4"   "base"
mgcv       "mgcv"       "/usr/lib/R/library" "1.8-23"  "recommended"
nlme       "nlme"       "/usr/lib/R/library" "3.1-131" "recommended"
nnet       "nnet"       "/usr/lib/R/library" "7.3-12"  "recommended"
parallel   "parallel"   "/usr/lib/R/library" "3.4.4"   "base"
rpart      "rpart"      "/usr/lib/R/library" "4.1-13"  "recommended"
spatial    "spatial"    "/usr/lib/R/library" "7.3-11"  "recommended"
splines    "splines"    "/usr/lib/R/library" "3.4.4"   "base"
stats      "stats"      "/usr/lib/R/library" "3.4.4"   "base"
stats4     "stats4"     "/usr/lib/R/library" "3.4.4"   "base"
survival   "survival"   "/usr/lib/R/library" "2.41-3"  "recommended"
tcltk      "tcltk"      "/usr/lib/R/library" "3.4.4"   "base"
```

```
tools      "tools"      "/usr/lib/R/library" "3.4.4"   "base"
utils      "utils"      "/usr/lib/R/library" "3.4.4"   "base"
           Depends
base       NA
boot       "R (>= 3.0.0), graphics, stats"
class      "R (>= 3.0.0), stats, utils"
cluster    "R (>= 3.0.1)"
codetools  "R (>= 2.1)"
compiler   NA
datasets   NA
foreign    "R (>= 3.0.0)"
graphics   NA
grDevices  NA
grid       NA
KernSmooth "R (>= 2.5.0), stats"
lattice    "R (>= 3.0.0)"
MASS       "R (>= 3.1.0), grDevices, graphics, stats, utils"
Matrix     "R (>= 3.0.1)"
methods    NA
mgcv       "R (>= 2.14.0), nlme (>= 3.1-64)"
nlme       "R (>= 3.0.2)"
nnet       "R (>= 2.14.0), stats, utils"
parallel   NA
rpart      "R (>= 2.15.0), graphics, stats, grDevices"
spatial    "R (>= 3.0.0), graphics, stats, utils"
splines    NA
stats      NA
stats4     NA
survival   "R (>= 2.13.0)"
tcltk      NA
tools      NA
utils      NA
           Imports                                                 LinkingTo
base       NA                                                      NA
boot       NA                                                      NA
class      "MASS"                                                  NA
cluster    "graphics, grDevices, stats, utils"                     NA
codetools  NA                                                      NA
compiler   NA                                                      NA
datasets   NA                                                      NA
foreign    "methods, utils, stats"                                 NA
graphics   "grDevices"                                             NA
grDevices  NA                                                      NA
grid       "grDevices, utils"                                      NA
KernSmooth NA                                                      NA
lattice    "grid, grDevices, graphics, stats, utils"               NA
MASS       "methods"                                               NA
Matrix     "methods, graphics, grid, stats, utils, lattice"        NA
methods    "utils, stats"                                          NA
mgcv       "methods, stats, graphics, Matrix"                      NA
nlme       "graphics, stats, utils, lattice"                       NA
nnet       NA                                                      NA
parallel   "tools, compiler"                                       NA
rpart      NA                                                      NA
spatial    NA                                                      NA
splines    "graphics, stats"                                       NA
stats      "utils, grDevices, graphics"                            NA
stats4     "graphics, methods, stats"                              NA
survival   "graphics, Matrix, methods, splines, stats, utils"      NA
tcltk      "utils"                                                 NA
tools      NA                                                      NA
utils      NA                                                      NA
           Suggests
base       "methods"
boot       "MASS, survival"
class      NA
cluster    "MASS"
```

```
codetools  NA
compiler   NA
datasets   NA
foreign    NA
graphics   NA
grDevices  "KernSmooth"
grid       "lattice"
KernSmooth "MASS"
lattice    "KernSmooth, MASS, latticeExtra"
MASS       "lattice, nlme, nnet, survival"
Matrix     "expm, MASS"
methods    "codetools"
mgcv       "splines, parallel, survival, MASS"
nlme       "Hmisc, MASS"
nnet       "MASS"
parallel   "methods"
rpart      "survival"
spatial    "MASS"
splines    "Matrix, methods"
stats      "MASS, Matrix, SuppDists, methods, stats4"
stats4     NA
survival   NA
tcltk      NA
tools      "codetools, methods, xml2, curl"
utils      "methods, XML"
           Enhances                                  License
base       NA                                        "Part of R 3.4.4"
boot       NA                                        "Unlimited"
class      NA                                        "GPL-2 | GPL-3"
cluster    NA                                        "GPL (>= 2)"
codetools  NA                                        "GPL"
compiler   NA                                        "Part of R 3.4.4"
datasets   NA                                        "Part of R 3.4.4"
foreign    NA                                        "GPL (>= 2)"
graphics   NA                                        "Part of R 3.4.4"
grDevices  NA                                        "Part of R 3.4.4"
grid       NA                                        "Part of R 3.4.4"
KernSmooth NA                                        "Unlimited"
lattice    "chron"                                   "GPL (>= 2)"
MASS       NA                                        "GPL-2 | GPL-3"
Matrix     "MatrixModels, graph, SparseM, sfsmisc"   "GPL (>= 2) | file LICENCE"
methods    NA                                        "Part of R 3.4.4"
mgcv       NA                                        "GPL (>= 2)"
nlme       NA                                        "GPL (>= 2) | file LICENCE"
nnet       NA                                        "GPL-2 | GPL-3"
parallel   "snow, nws, Rmpi"                         "Part of R 3.4.4"
rpart      NA                                        "GPL-2 | GPL-3"
spatial    NA                                        "GPL-2 | GPL-3"
splines    NA                                        "Part of R 3.4.4"
stats      NA                                        "Part of R 3.4.4"
stats4     NA                                        "Part of R 3.4.4"
survival   NA                                        "LGPL (>= 2)"
tcltk      NA                                        "Part of R 3.4.4"
tools      NA                                        "Part of R 3.4.4"
utils      NA                                        "Part of R 3.4.4"
           License_is_FOSS License_restricts_use OS_type MD5sum
base       NA              NA                    NA      NA
boot       NA              NA                    NA      NA
class      NA              NA                    NA      NA
cluster    NA              NA                    NA      NA
codetools  NA              NA                    NA      NA
compiler   NA              NA                    NA      NA
datasets   NA              NA                    NA      NA
foreign    NA              NA                    NA      NA
graphics   NA              NA                    NA      NA
grDevices  NA              NA                    NA      NA
grid       NA              NA                    NA      NA
```

```
KernSmooth NA              NA                    NA      NA
lattice    NA              NA                    NA      NA
MASS       NA              NA                    NA      NA
Matrix     NA              NA                    NA      NA
methods    NA              NA                    NA      NA
mgcv       NA              NA                    NA      NA
nlme       NA              NA                    NA      NA
nnet       NA              NA                    NA      NA
parallel   NA              NA                    NA      NA
rpart      NA              NA                    NA      NA
spatial    NA              NA                    NA      NA
splines    NA              NA                    NA      NA
stats      NA              NA                    NA      NA
stats4     NA              NA                    NA      NA
survival   NA              NA                    NA      NA
tcltk      NA              NA                    NA      NA
tools      NA              NA                    NA      NA
utils      NA              NA                    NA      NA
           NeedsCompilation Built
base       NA               "3.4.4"
boot       "no"             "3.4.2"
class      "yes"            "3.4.2"
cluster    "yes"            "3.4.2"
codetools  "no"             "3.4.2"
compiler   NA               "3.4.4"
datasets   NA               "3.4.4"
foreign    "yes"            "3.4.4"
graphics   "yes"            "3.4.4"
grDevices  "yes"            "3.4.4"
grid       "yes"            "3.4.4"
KernSmooth "yes"            "3.4.2"
lattice    "yes"            "3.4.2"
MASS       "yes"            "3.4.3"
Matrix     "yes"            "3.4.2"
methods    "yes"            "3.4.4"
mgcv       "yes"            "3.4.3"
nlme       "yes"            "3.4.2"
nnet       "yes"            "3.4.2"
parallel   "yes"            "3.4.4"
rpart      "yes"            "3.4.3"
spatial    "yes"            "3.4.2"
splines    "yes"            "3.4.4"
stats      "yes"            "3.4.4"
stats4     NA               "3.4.4"
survival   "yes"            "3.4.2"
tcltk      "yes"            "3.4.4"
tools      "yes"            "3.4.4"
utils      "yes"            "3.4.4"
```

6. Citing R in papers and publications

Credit where credit is due, when using *R* or any of its packages it is important to credit the work of the developers. *R* has a built-in function citation() to get the BibTex entry for reference management software or simply take the reference directly from the terminal as text.

6.1 Cite R

```
> citation()
  R Core Team (2018). R: A language and environment for statistical
  computing. R Foundation for Statistical Computing, Vienna, Austria.
  URL https://www.R-project.org/.

A BibTeX entry for LaTeX users is

  @Manual{,
    title = {R: A Language and Environment for Statistical Computing},
    author = {{R Core Team}},
    organization = {R Foundation for Statistical Computing},
    address = {Vienna, Austria},
    year = {2018},
    url = {https://www.R-project.org/},
  }

We have invested a lot of time and effort in creating R, please cite it
when using it for data analysis. See also 'citation("pkgname")' for
citing R packages.
```

6.2 Cite individual R packages

```
> citation(package = "lme4")

To cite lme4 in publications use:

  Douglas Bates, Martin Maechler, Ben Bolker, Steve Walker (2015).
  Fitting Linear Mixed-Effects Models Using lme4. Journal of
  Statistical Software, 67(1), 1-48. doi:10.18637/jss.v067.i01.

A BibTeX entry for LaTeX users is

  @Article{,
    title = {Fitting Linear Mixed-Effects Models Using {lme4}},
    author = {Douglas Bates and Martin M{\"a}chler and Ben Bolker and Steve Walker},
    journal = {Journal of Statistical Software},
    year = {2015},
    volume = {67},
    number = {1},
    pages = {1--48},
    doi = {10.18637/jss.v067.i01},
  }
```

7. Vectors, the basic data structure in R

The *vector* is the basic data structure in *R*. It contains element of the same type. The data types can be logical, integer, double, character, complex or raw. It is considered the fundamental data type in *R*.

- **logical** - TRUE and FALSE are reserved words denoting logical constants, whereas T and F are global variables whose initial values set to these.
- **integer** - Whole number (not a fraction) that can be positive, negative, or zero.
- **double** - Creates a double-precision vector of the specified length. The elements of the vector are all equal to 0. It is identical to numeric.
- **character** - Type of indexing is useful when dealing with named vectors.
- **complex** - The vector can be specified either by giving its length, its real and imaginary parts, or modulus and argument.
- **raw** - Type is intended to hold raw bytes. It is possible to extract sub-sequences of bytes, and to replace elements.

```
> x = 3

> y = 5

> meatballs = 7

> meatballs + x - y
[1] 5

> 4 + 3
[1] 7

> a = 4 + 3
> a
[1] 7
```

The vector is the fundamental data type in *R*.

e.g. **[1] 1 2 3 4 5 6 7**

Numbers being put together in a vector must be done in one of these four ways.

```
> x = c(1,2,3,4,5,6,7)  # concatenate
> y = 1:7               # colon operator
> z = seq(1,7,1)        # sequence, 1 to 7 in intervals of 2 i.e. 1 3 5 7
> m = rep(1:7,2)        # repeat i.e. 1,2,3,4,5,6,7,1,2,3,4,5,6,7
```

Now consider each.

```
> x = c(1,2,3,4,5,6,7)
> y = 1:7
> z = seq(1,7,1)
> m = rep(1:7,2)

> x
[1] 1 2 3 4 5 6 7
> y
[1] 1 2 3 4 5 6 7
> z
[1] 1 2 3 4 5 6 7
> m
[1] 1 2 3 4 5 6 7
```

Try some variations.

```
> d = seq(1,12,3)
> d
[1]  1  4  7 10

> e = rep(1:4,3)
> e
 [1] 1 2 3 4 1 2 3 4 1 2 3 4

> f = rep(d,4)
> f
 [1]  1  4  7 10  1  4  7 10  1  4  7 10  1  4  7 10
```

Note:

- **c(...):** - Combines or concatenates its arguments.
- **seq(from, to, by=):** - Generate regular sequences.
- **rep(x, ...):** - replicates the values in 'x'.

Combining these.

```
> x = 1:7

> y = seq(1,7,2)

> z = c(x,y)

> z
 [1] 1 2 3 4 5 6 7 1 3 5 7
```

Any time more than one number must be given to *R*, then it MUST be created using one of these functions.

7.1 Exercise: Generate vectors

```
5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

1.0 1.3 1.6 1.9 2.2 2.5 2.8 3.1 3.4 3.7 4.0 4.3 4.6 4.9

7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7

14 12 10  8  6  4  2  0

 5 7 6 10 4 3 17

1 2 3 4 1 2 3 4 1 2 3 4

1 2 3 4 1 2 3 4 1 2 3 4 85

2.00 2.17 2.34 2.51 2.68 2.85 3.02 3.19 3.36 3.53 3.70 3.87 4.04 4.21 4.38 4.55 4.72
4.89 5.06 5.23 5.40 5.57 5.74 5.91 6.08

"bird" "cat" "ferret"

5 5 12 12 13 13 20 20

5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

1.0 1.3 1.6 1.9 2.2 2.5 2.8 3.1 3.4 3.7 4.0 4.3 4.6 4.9
```

Answer:

```
5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
> a = 5:23

1.0 1.3 1.6 1.9 2.2 2.5 2.8 3.1 3.4 3.7 4.0 4.3 4.6 4.9
> c = seq(1,4.9,0.3)

7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
> c = rep(7, 18)

14 12 10  8  6  4  2  0
> c = seq(14,0,-2)

 5 7 6 10 4 3 17
> c = c(5,7,6,10,4,3,17)

1 2 3 4 1 2 3 4 1 2 3 4
> rep(1:4,3)

1 2 3 4 1 2 3 4 1 2 3 4 85
> c = c(rep(1:4,3), 85)

2.00 2.17 2.34 2.51 2.68 2.85 3.02 3.19 3.36 3.53 3.70 3.87 4.04 4.21 4.38 4.55 4.72
4.89 5.06 5.23 5.40 5.57 5.74 5.91 6.08
> c = seq(2,length.out=25,by=0.17)

"bird" "cat" "ferret"
> c = c("bird","cat","ferret")

5 5 12 12 13 13 20 20
> c = rep(c(5,12,13,20),each=2)
```

7.2 Length() of a vector

The length of a vector can be obtained by the *length()* function.

```
> c = (seq(1, 1000, 0.34))
> length(c)
[1] 2939

> c = length(seq(1, 1000, 0.34))
> c
[1] 2939
```

7.3 Vector principles

3 vector principles central to R programming.

1. Recycling
2. Vectorisation
3. Filtering [indexing]

7.3.1 Recycling

When applying an operation to two vectors that requires them to be the same length, *R* automatically recycles or repeats the shorter vector until it is the same length as the longer one.

```
> x = c(1,2,3)
> y = c(3,4,5)
> z = 15
> a = c(8,9)
> b = c(10,20)
```

Looking at *x+y*, *1+3=4*, *2+4=6* and *3+5=8*.

```
> x+y
[1] 4 6 8
```

Now looking at *x+a*, *1+8=9*, *2+9=11* and *3+?*. As *a* has ran out of elements the first element is recycled again so *3+8=11*. The following warning message is received.

```
> x+a
[1]  9 11 11
Warning message:
In x + a : longer object length is not a multiple of shorter object length
```

Now looking at *y*b*, *3*10=30*, *4*20=80* and as *a* has ran out of elements the first is recycled again so *5*10=50* and a warning message is received.

```
> y*b
[1] 30 80 50
Warning message:
In y * b : longer object length is not a multiple of shorter object length
```

And finally *y*z*, *3*15=45* and as there are no more elements in *z*, *15* must be recycled and the final calculations are *4*15=60* and *5*15=75*.

```
> y*z
[1] 45 60 75
Warning message:
In y * b : longer object length is not a multiple of shorter object length
```

The last example above is something that happens all the time, where all the elements in *y = c(3,4,5)* are multiplied by the single element in *b=15*.

- 3 x 15 = 45
- 4 x (15 recycled) = 60
- 5 x (15 recycled) = 75

7.3.2 Vectorisation

Scalars are vectors, therefore most functions that you can apply to a single value, you can apply to a vector of values.

```
> sqrt(16)
[1] 4

> x = c(3,6,9,12,15)
> sqrt(x)
[1] 1.732051 2.449490 3.000000 3.464102 3.872983

> sqrt(x/2)
[1] 1.224745 1.732051 2.121320 2.449490 2.738613

> x^2
[1]   9  36  81 144 225

> x/3
[1] 1 2 3 4 5
```

Note here the *modulus*.

```
> x %% 4        # modulus (x mod y) 5%%2 is 1
[1] 3 2 1 0 3
```

Breaking this down further.

```
> 3 %% 4
[1] 3
> 6 %% 4
[1] 2
> 9 %% 4
[1] 1
> 12 %% 4
[1] 0
> 15 %% 4
[1] 3
```

Looking at a different modulus example, *4* and *5* are continuously recycled until all the elements of *x* are calculated.

```
> x %% c(4,5)
[1] 3 1 1 2 3
Warning message:
In x%%c(4, 5) :
  longer object length is not a multiple of shorter object length
```

Exercise: write code to produce this vector

1 4 9 16 25 36 49 64 81 100

```
> a = seq(1,10,1)
> a^2
 [1]   1   4   9  16  25  36  49  64  81 100

> seq(1,10,1)^2
 [1]   1   4   9  16  25  36  49  64  81 100

> (1:10)^2
 [1]   1   4   9  16  25  36  49  64  81 100
```

Note the importance of the brackets. Leaving out the brackets gives a completely different answer.

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

7.3.3 Filtering [Indexing] of vectors

vector1[*vector2*]

The elements of *vector2* select the elements of *vector1* via the [ ] brackets.

```
> y = c(10,1,16,3,8,5,13,55,34,13)
```

Access the third element of the concatenation *y*.

```
> y[3]
[1] 16
```

Now access the first four elements of the concatenation *y*.

```
> y[1:4]
[1] 10  1 16  3
```

So what happens for *y[3,4,6,8]*?

```
> y[3,4,6,8]
Error in y[3, 4, 6, 8] : incorrect number of dimensions
```

Well *[3,4,6,8]* is not a vector. Remember that a vector must be developed as concatenation, via a colon operator, a sequence or a repeat. So rewrite as:

```
> y[c(3,4,6,8)]
[1] 16  3  5 55
```

So what happened?, remember *y = c(10,1,16,3,8,5,13,55,34,13)*. The second vector selects elements from the first vector. *3* selects the third element of *y* which is *16*, *4* selects the forth element *3*, etc..

Exercise (a)

Extract the 3rd, 4th and 7th numbers in this vector.

y = c(1,1,2,3,5,8,13,21,34,55)

```
> y = c(1,1,2,3,5,8,13,21,34,55)

> x = c(3,4,7)
> y[x]
[1]  2  3 13

> y[c(3,4,7)]
[1]  2  3 13
```

Exercise (b)

Extract the first 6 numbers of the vector.

```
> y = c(1,1,2,3,5,8,13,21,34,55)

> x = 1:6
> y[x]
[1] 1 1 2 3 5 8

> y[1:6]
[1] 1 1 2 3 5 8
```

Exercise (c)

Extract the final number of this vector by using the *length()* function.

```
> y = c(1,1,2,3,5,8,13,21,34,55)

> y[length(y)]
[1] 55
```

Another way to achieve the same result is to:

```
> max(y)
[1] 55
```

7.3.4 Other Filtering [Indexing] examples

```
> y = c(7,8,3,2,4,5,6,3,4,5)

> min(y)             # Returns the smallest element
[1] 2

> max(y)             # Returns the largest element
[1] 8

> y[2:4]             # Returns elements 2 - 4
[1] 8 3 2

> y[3:6]             # Returns elements 3 - 6
[1] 3 2 4 5
```

7.3.5 Ordering elements in a vector

The *order()* function returns a permutation which rearranges its first argument into ascending or descending order, breaking ties by further arguments. As can be seen from the example below it returns the positions of the elements in *y* based on the sequential size starting with position *2* which is *1*, then position *4* for the next lowest number *3*, etc..

```
> y = c(10,1,16,3,8,5,13,55,34,13)
> order(y)
 [1]  2  4  6  5  1  7 10  3  9  8
```

If the actual ordered list of the elements instead of the relevant element positions is required then use the following.

```
> y[order(y)]
 [1]  1  3  5  8 10 13 13 16 34 55
```

Reversing the order. The *rev()* function provides a reversed version of its argument or alternatively negate the *y* in the order function.

```
> y[rev(order(y))]
 [1] 55 34 16 13 13 10  8  5  3  1

> y[order(-y)]
 [1] 55 34 16 13 13 10  8  5  3  1
```

7.3.6 Extracting elements from a vector

How do I extract numbers from a vector?

- Write down the name of the vector.
- Put square brackets after it.
- Put something inside the square brackets.

```
> y = c(10,1,16,3,8,5,13,55,34,13)

> y[5]
[1] 8

> y[7]
[1] 13
```

What if one wants to extract certain numbers?

All numbers > 7.

```
> y = c(10,1,16,3,8,5,13,55,34,13)

> y[y < 7]
[1] 1 3 5
```

All the numbers from 1 to 6.

```
> y[c(1,2,3,4,5,6)]
[1] 10  1 16  3  8  5

> y[1:6]
[1] 10  1 16  3  8  5
```

Only numbers equal to 3.

```
> y = c(10,1,16,3,8,5,13,55,34,13)

> y[y = 3]
[1] 16
```

Using negatives to eliminate numbers from a list.

```
> y = c(10,1,16,3,8,5,13,55,34,13)

> y[c(-2,-2)]
[1] 10 16  3  8  5 13 55 34 13

> y[-1:-4]
[1]  8  5 13 55 34 13
```

7.4 Booleans

Returns True or False (1 or 0).

```
> y = c(1,1,2,3,5,8,13,21,34,55)

> y == 3
 [1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE

> y <= 13
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE

> y >= 8
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE

> y > 8
 [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE

> y != 8
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE

> y^2 < 8
 [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

7.4.1 Exercise

What happens for: *mean(y==3)*?

Note: Generic function for the (trimmed) arithmetic mean.

Well the mean of *y*, in other words *sum(5)/length(5)*.

```
> mean(y)
[1] 14.3

> sum(y)/length(y)
[1] 14.3
```

'==' means exactly equal to. Remember the output of *y == 3*?

```
> y == 3
 [1] FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE
```

So that is *0,0,0,1,0,0,0,0,0,0*.

```
> sum(c(0,0,0,1,0,0,0,0,0,0)/10)
[1] 0.1
```

It is the *mean* of the booleans returned from the logic *y==3* statement or a single one in 10 elements.

```
> mean(y==3)
[1] 0.1
```

Now put the logic boolean statements inside the square brackets.

- T (TRUE) = include
- F (FALSE) = exclude

```
> y = c(1,1,2,3,5,8,13,21,34,55)

> y[y <= 10]
[1] 1 1 2 3 5 8

> y[y == 3]
[1] 3

> y[y <=10]
[1] 1 1 2 3 5 8

> y[y >= 9]
[1] 13 21 34 55

> y[y > 9]
[1] 13 21 34 55

> y[y != 8]
[1]  1  1  2  3  5 13 21 34 55

> y[y^2 < 10]
[1] 1 1 2 3
```

or map True (T) and False (F) to the values in the vector.

```
> a = c(1,3,5,7,9)

> a[c(T,F,F,T,F)]
[1] 1 7
```

In this case there are not enough boolean statements so they are recycled such that it is the same as: *a[c(T,F,T,F,T)]*.

```
> a[c(T,F)]
[1] 1 5 9

> a[c(T,F,T,F,T)]
[1] 1 5 9
```

7.4.2 Combining Booleans (and (&), or (|))

```
> x = c(3,6,9,12,15)
```

Both conditions must be True.

```
> x > 3 & x < 10
[1] FALSE  TRUE  TRUE FALSE FALSE
```

Either condition is True.

```
> x == 12 | x < 5
[1]  TRUE FALSE FALSE  TRUE FALSE
```

7.4.3 Boolean operators

==	equals
<	is less than
>	is greater than
<=	less than or equal to
>=	greater than or equal to
!=	is not equal to
&	and
|	or

Exercise 1

For the vector *f = c(1,2,3,6,10,15,21,25,29,30)*

1. Find all numbers equal to 15.
2. Find all numbers greater than 9
3. Find all numbers not equal to 10
4. Find 6th to 10th vector elements
5. Find all except the final element
6. Find all numbers that are multiples of 5
7. Find all numbers less than or equal to 15
8. Find all numbers between 7 and 24

Answer:

```
# Find all numbers equal to 15.
> f[f==15]
[1] 15

# Find all numbers greater than 9.
> f[f > 9]
[1] 10 15 21 25 29 30

# Find all numbers not equal to 10.
> f[f != 10]
[1]  1  2  3  6 15 21 25 29 30

# Find 6th to 10th vector elements.
> f[f = 6:10]
[1] 15 21 25 29 30

# Find all except the final element.
> f[-length(f)]
[1]  1  2  3  6 10 15 21 25 29

# Find all numbers that are multiples of 5.
> f[(f %% 5) == 0]
[1] 10 15 25 30

# Find all numbers less than or equal to 15.
> f[f <= 15]
[1]  1  2  3  6 10 15

# Find all numbers between 7 and 24.
> f[(f > 7) & (f < 24)]
[1] 10 15 21
```

Exercise 2

For the vector *f = c(1,2,3,6,10,15,21,25,29,30)*

1. Find all numbers greater than 9
2. Find all numbers not equal to 10
3. Find 6th to 10th vector elements
4. Find all except the final element
5. Find all numbers that are multiples of 5
6. Find all numbers less than or equal to 15
7. Find all numbers between 7 and 24

Answer:

```
> f = c(1,2,3,6,10,15,21,25,29,30)

# Find all numbers greater than 9
> f[f > 9]
[1] 10 15 21 25 29 30

# Find all numbers not equal to 10
> f[f != 9]
 [1]  1  2  3  6 10 15 21 25 29 30

# Find 6th to 10th vector elements
> f[6:10]
[1] 15 21 25 29 30

# Find all except the final element
> f[-length(f)]
[1]  1  2  3  6 10 15 21 25 29

# Find all numbers that are multiples of 5
> f[f %% 5 == 0]
[1] 10 15 25 30

# Find all numbers less than or equal to 15
> f[f <= 15]
[1]  1  2  3  6 10 15

# Find all numbers between 7 and 24
> f[(f > 7) & (f < 16)]
[1] 10 15
```

8. Building on Vectors, other data structures in R

8.1 Basics of structures

vector[*row*]

matrix[*row,column*]

array[*row,column,level*]

8.1.1 Vectors

- The vector is the fundamental data type in *R*, scalars and matrices are just special types of vectors and that all things that apply to vectors, apply to these.

8.1.2 Matrix

- A matrix is a collection of data elements arranged in a two-dimensional rectangular layout. The following is an example of a matrix with 2 rows and 3 columns.
- The elements must be of the same type.
- Here is an example.

```
> a.matrix = matrix(c(2, 4, 3, 1, 5, 7), # the data elements
                    nrow=2,              # number of rows
                    ncol=3,              # number of columns
                   )

> a.matrix
     [,1] [,2] [,3]
[1,]    2    3    5
[2,]    4    1    7

> a.matrix = matrix(c(2, 4, 3, 1, 5, 7), # the data elements
                    nrow=2,              # number of rows
                    ncol=3,              # number of columns
                    byrow = TRUE         # fill matrix by rows
                   )

> a.matrix
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    1    5    7
```

- Indexing Matrices

matrix.name[*Row,Column*]

- So to access row 2 and column 3 - *matrix.name[2,3]* or to access all elements in row 2 - *matrix.name[2,]* or all the elements in column 3 - *matrix.name[,3]*.

```
> a.matrix[2,3]
[1] 7

> a.matrix[2,]
[1] 1 5 7
```

```
> a.matrix[,3]
[1] 3 7

> a.matrix[1:2,2:3]
     [,1] [,2]
[1,]    4    3
[2,]    5    7

> a.matrix[-2,]
[1] 2 4 3

> a.matrix[,-2]
     [,1] [,2]
[1,]    2    3
[2,]    1    7
```

- How about finding all the rows where column 3 is greater than 6?

```
> a.matrix [a.matrix[,3] > 6,]
[1] 1 5 7
```

Apply boolean questions to matrix

For a matrix of four rows and three columns with the data *1,5,9,2,6,10,3,7,11,4,8,12* get the output of the following *R* commands.

- z.matrix == 3
- z.matrix[,3] <= 10
- z.matrix != 3
- any(z.matrix[1,] == 3)
- any(z.matrix > 15)
- all(z.matrix == 3)
- all(z.matrix < 20)
- complete.cases(z.matrix)

```
> z.matrix = matrix(c(1,5,9,2,6,10,3,7,11,4,8,12),
                    nrow=4,
                    ncol=3,
                    byrow=TRUE
                   )

> z.matrix
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

> z.matrix == 3
      [,1]  [,2]  [,3]
[1,] FALSE FALSE FALSE
[2,] FALSE FALSE FALSE
[3,]  TRUE FALSE FALSE
[4,] FALSE FALSE FALSE

> z.matrix[,3] <= 10
[1]  TRUE  TRUE FALSE FALSE
```

```
> z.matrix != 3
      [,1] [,2] [,3]
[1,]  TRUE TRUE TRUE
[2,]  TRUE TRUE TRUE
[3,] FALSE TRUE TRUE
[4,]  TRUE TRUE TRUE

> any(z.matrix[1,] == 3)
[1] FALSE

> any(z.matrix > 15)
[1] FALSE

> all(z.matrix == 3)
[1] FALSE

> all(z.matrix < 20)
[1] TRUE

> complete.cases(z.matrix)
[1] TRUE TRUE TRUE TRUE
```

8.1.3 Array

Arrays are the R data objects which can store data in more than two dimensions. An array is created using the *array()* function. It takes vectors as input and uses the values in the *dim* (dimensions) parameter to create an array. The *dim* parameters define matrices of three rows, four columns and two deep, i.e. two matrices.

array[*row,column,level*]

Here is an example.

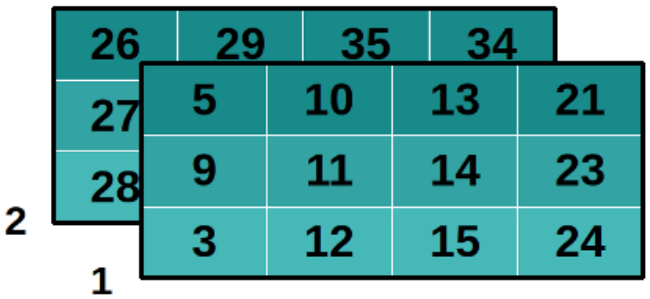


Illustration 3: Array

```
> v1 = c(5,9,3)
> v2 = c(10,11,12,13,14,15)
> v3 = c(21,23,24,26,27,28,29)
> v4 = c(32,34,35,37,31,34,45,46)
> p = array(c(v1,v2,v3,v4),dim = c(3,4,2))
```

```
> p

, , 1

     [,1] [,2] [,3] [,4]
[1,]    5   10   13   21
[2,]    9   11   14   23
[3,]    3   12   15   24

, , 2

     [,1] [,2] [,3] [,4]
[1,]   26   29   35   34
[2,]   27   32   37   45
[3,]   28   34   31   46
```

- Here is another example.

```
> v1 = c(5,9,3)
> v2 = c(10,11,12,13,14,15)
> p = array(c(v1,v2),dim = c(3,3,2))
> p
, , 1

     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15

, , 2

     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15
```

Names can be given to the rows, columns and matrices in the array by using the *dimnames* parameter.

Here is example.

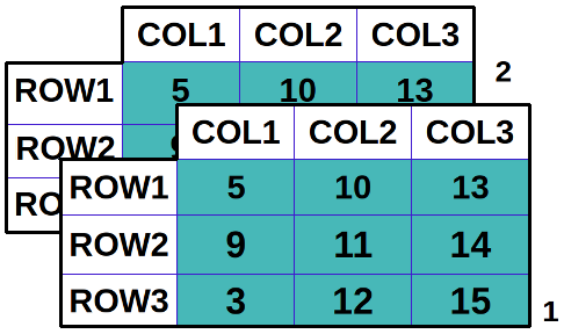


Illustration 4: Array 2

```
> v1 = c(5,9,3)

> v2 = c(10,11,12,13,14,15)

> column.names = c("COL1","COL2","COL3")

> row.names = c("ROW1","ROW2","ROW3")

> matrix.names = c("Matrix1","Matrix2")

> p = array(c(v1,v2),dim = c(3,3,2),dimnames =
list(row.names,column.names,matrix.names))

> p
, , Matrix1

     COL1 COL2 COL3
ROW1    5   10   13
ROW2    9   11   14
ROW3    3   12   15

, , Matrix2

     COL1 COL2 COL3
ROW1    5   10   13
ROW2    9   11   14
ROW3    3   12   15
```

Exercise (a): Matrix

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]   11   13   15   17   19
[3,]   21   23   25   27   29
[4,]   31   33   35   37   39
```

Create a matrix like the one above, containing the sequence of odd numbers starting at 1 and counting up to fill the matrix.

Answer:

```
> new.matrix = matrix(seq(1,length.out=20,by=2),
                      nrow=4,ncol=5,
                      byrow=TRUE
                     )

> new.matrix
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]   11   13   15   17   19
[3,]   21   23   25   27   29
[4,]   31   33   35   37   39
```

Exercise (b): Matrix

1. Create a matrix (called d1) with 6 rows and 4 columns (*byrow=F*) using the sequence of odd numbers starting at 1.
2. Extract the number from the 3rd column, 4th row and assign it to the variable name g1.
3. Extract the 6th row
4. Extract columns 2 & 4 from d1 and call the new matrix d2
5. Create a new matrix (d3), by doing something to d1, that contains the sequence of EVEN numbers starting at 2
6. Change d3 element 3rd row, 2nd column so that it equals 500. Look at *d3*. Change 3rd row, 2nd column to *Harry*. Look at *d3* again. Has anything changed? Why? Change *Harry* back to *500*. Now what has happened?

```
> d1 = matrix(seq(1,length.out=24,by=2),
              nrow=6,ncol=4,
              byrow=FALSE
             )

> d1
     [,1] [,2] [,3] [,4]
[1,]    1   13   25   37
[2,]    3   15   27   39
[3,]    5   17   29   41
[4,]    7   19   31   43
[5,]    9   21   33   45
[6,]   11   23   35   47

> g1 = d1[4,3]

> g1
[1] 31

> d1[6,]
[1] 11 23 35 47

> c(d1[,2], d1[,4])
[1] 13 15 17 19 21 23 37 39 41 43 45 4

> d2 = matrix(c(d1[,2], d1[,4]),
              nrow=6,ncol=2,
              byrow=FALSE
             )

> d3 = d1 + 1

> d3
     [,1] [,2] [,3] [,4]
[1,]    2   14   26   38
[2,]    4   16   28   40
[3,]    6   18   30   42
[4,]    8   20   32   44
[5,]   10   22   34   46
[6,]   12   24   36   48

> d3[3,2] = 500
```

```
> d3
     [,1] [,2] [,3] [,4]
[1,]    2   14   26   38
[2,]    4   16   28   40
[3,]    6  500   30   42
[4,]    8   20   32   44
[5,]   10   22   34   46
[6,]   12   24   36   48

> d3[3,2] = 'Harry'

> d3
     [,1] [,2]    [,3] [,4]
[1,] "2"  "14"    "26" "38"
[2,] "4"  "16"    "28" "40"
[3,] "6"  "Harry" "30" "42"
[4,] "8"  "20"    "32" "44"
[5,] "10" "22"    "34" "46"
[6,] "12" "24"    "36" "48"
```

All values became strings because in a matrix all values must be of the same type.

```
> d3[3,2] = 500

> d3
     [,1] [,2]  [,3] [,4]
[1,] "2"  "14"  "26" "38"
[2,] "4"  "16"  "28" "40"
[3,] "6"  "500" "30" "42"
[4,] "8"  "20"  "32" "44"
[5,] "10" "22"  "34" "46"
[6,] "12" "24"  "36" "48"

String type has been maintained.
```

Exercise (c): Matrix filtering

```
> chick = matrix(c(seq(1,5,1),
                   c(10,15,12,13,15,8,11,9,12,13)),
                   nrow=5,ncol=3,byrow=FALSE
                  )

> colnames(chick) = c('Individual','Weight','Age')

> chick
     Individual Weight Age
[1,]          1     10   8
[2,]          2     15  11
[3,]          3     12   9
[4,]          4     13  12
[5,]          5     15  13
```

From the *chick* matrix:

1. All rows where *weight* is less than 15g.
2. Using the mean() function, calculate the mean age of chicks *>10g* in weight.
3. Add an extra column (2,5,3,5,6) to the *chick* matrix using the *cbind()* function.

cbind(): - Take a sequence of vector, matrix or data-frame arguments and combine by columns or rows, respectively.

```
> chick[chick[,2] < 15,]
     Individual Weight Age
[1,]          1     10   8
[2,]          3     12   9
[3,]          4     13  12

> mean(chick[chick[,2] > 10,'Age'])
[1] 11.25

> chick = cbind(chick,c(2,5,3,5,6))

> chick
     Individual Weight Age
[1,]          1     10   8 2
[2,]          2     15  11 5
[3,]          3     12   9 3
[4,]          4     13  12 5
[5,]          5     15  13 6
```

8.1.4 The 'apply' family of functions

apply(): returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.

apply(*X, MARGIN, FUN, ...*)

- ***X*:** an array, including a matrix.
- ***MARGIN*:** 1 = rows, 2 = columns, c(1,2) = both.
- ***FUN*:** the function.

```
> v1 = c(5,9,3)
> v2 = c(10,11,12,13,14,15,16,17,18)
> v3 = c(21,23,24,26,27,28,29,30)
> p = matrix(c(v1,v2,v3), nrow = 4, ncol = 5)

> p
     [,1] [,2] [,3] [,4] [,5]
[1,]    5   11   15   21   27
[2,]    9   12   16   23   28
[3,]    3   13   17   24   29
[4,]   10   14   18   26   30

# Sum of each column
> apply(p, 2, sum)
[1]  27  50  66  94 114

# Get the mean of each row
> apply(p, 1, mean)
[1] 15.8 17.6 17.2 19.6

> apply(p, c(1,2), mean)
     [,1] [,2] [,3] [,4] [,5]
[1,]    5   11   15   21   27
[2,]    9   12   16   23   28
[3,]    3   13   17   24   29
[4,]   10   14   18   26   30
```

While it is possible to supply a vector *c(1,2)* to the *MARGIN* argument it makes little sense with a *matrix*. However with an *array* it operates between the matrices within the *array*.

```
> v1 = c(5,9,3)

> v2 = c(10,11,12,13,14,15)

> v3 = c(21,23,24,26,27,28,29)

> v4 = c(32,34,35,37,31,34,45,46)

> a = array(c(v1,v2,v3,v4),dim = c(3,4,2))

> a
, , 1

     [,1] [,2] [,3] [,4]
[1,]    5   10   13   21
[2,]    9   11   14   23
[3,]    3   12   15   24

, , 2

     [,1] [,2] [,3] [,4]
[1,]   26   29   35   34
[2,]   27   32   37   45
[3,]   28   34   31   46


> apply(a, c(1,2), mean)
     [,1] [,2] [,3] [,4]
[1,] 15.5 19.5 24.0 27.5
[2,] 18.0 21.5 25.5 34.0
[3,] 15.5 23.0 23.0 35.0
```

Other apply() functions

Other apply() functions exists, some of which are described later.

- **lapply():** For lists and data-frames.
- **sapply():** A simplified wrapper function for *lapply()*.
- **sapply():** The multivariate apply which can vectorise arguments to a function that is not usually accepting vectors as arguments.

In short, *mapply()* applies a function to multiple list or multiple vector arguments.

8.2 Recap exercises

1. Create the following vectors:

```
3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48

3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

20 13 16 17 18 20

100  97  94  91  88  85  82  79  76  73  70  67  64  61

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

13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

1  3  5  7  9 11 13 15 17 19

"bird"    "fish"    "cricket"
```

- Answer

```
> seq(3, 48, 3)
 [1]  3  6  9 12 15 18 21 24 27 30 33 36 39 42 45 48

> rep(3,25)
 [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

> c(20,13,16,17,18,20)
[1] 20 13 16 17 18 20

> seq(100, 61, -3)
 [1] 100  97  94  91  88  85  82  79  76  73  70  67  64  61

> seq(1, 20, 1)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

> seq(13, 29, 1)
 [1] 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

> seq(1, 19, 2)
 [1]  1  3  5  7  9 11 13 15 17 19

> c('bird','fish','cricket')
[1] "bird"    "fish"    "cricket"
```

2. From the following vector

   3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48

   - Multiply each element by 10.
   - Take the square root of each element.
   - Add three to each element.
   - Standardise each element to the mean of the vector. (i.e. divide each element by the vector's mean).
   - Multiply every second element by the vector's length.
   - Answer

```
> d = seq(3, 48, 3)
> d
 [1]  3  6  9 12 15 18 21 24 27 30 33 36 39 42 45 48

> d * 10
 [1]  30  60  90 120 150 180 210 240 270 300 330 360 390 420 450 480

> sqrt(d)
 [1] 1.732051 2.449490 3.000000 3.464102 3.872983 4.242641 4.582576
 [8] 4.898979 5.196152 5.477226 5.744563 6.000000 6.244998 6.480741
 [15] 6.708204 6.928203

> d + 3
 [1]  6  9 12 15 18 21 24 27 30 33 36 39 42 45 48 51

> d/(mean(d))
 [1] 0.1176471 0.2352941 0.3529412 0.4705882 0.5882353 0.7058824
 [7] 0.8235294 0.9411765 1.0588235 1.1764706 1.2941176 1.4117647
 [15] 1.5294118 1.6470588 1.7647059 1.8823529

> d * c(1, length(d))
 [1]   3  96   9 192  15 288  21 384  27 480  33 576  39 672  45 768
```

3. Using vector indexing extract the following vectors from the *a* vector where:

```
a = c(5, 1, 6, 10, 2, 7, 13, 12, 8)

5 1 6

5 5 5 13 5 5 1 6 10 2 7 13 12

 1 2 5 7 8 10 12 13

numbers less than 10

numbers equal to 7

all even numbers
```

- Answer

```
> a[1:3]
[1] 5 1 6

> c(rep(a[1],3),a[7],a[1])
[1]  5  5  5 13  5

> a[1:length(a) - 1]
[1]  5  1  6 10  2  7 13 12

> c(a[2],a[5],a[1],a[3],a[6],a[9],a[4],a[8],a[7])
[1]  1  2  5  6  7  8 10 12 13

> a[a < 10]
[1] 5 1 6 2 7 8

> a[a == 7]
[1] 7

> a[a %% 2 == 0]
[1]  6 10  2 12  8
```

4. Create the following matrix

```
     [,1] [,2] [,3] [,4]
[1,]    2    4    6    8
[2,]   10   12   14   16
[3,]   18   20   22   24
[4,]   26   28   30   32
[5,]   34   36   38   40
[6,]   42   44   46   48
```

- Extract the 3rd column.
- Extract the 5th row.
- Create a new matrix from the first three rows and last two columns.
- Extract all numbers greater than 25.
- Extract all numbers greater than 35 from row five.
- Extract all numbers greater than 35 from column four.
- Extract all numbers between 10 & 30 from column one.

- Answer

```
> x.matrix = matrix(c(seq(2,48,2)),
                    nrow=6,ncol=4,byrow=TRUE
                   )

> x.matrix
     [,1] [,2] [,3] [,4]
[1,]    2    4    6    8
[2,]   10   12   14   16
[3,]   18   20   22   24
[4,]   26   28   30   32
[5,]   34   36   38   40
[6,]   42   44   46   48

> x.matrix[,3]
[1]  6 14 22 30 38 46

> x.matrix[5,]
[1] 34 36 38 40

> x.matrix[1:3,3:4]
     [,1] [,2]
[1,]    6    8
[2,]   14   16
[3,]   22   24

> x.matrix[c(x.matrix[]) > 25]
 [1] 26 34 42 28 36 44 30 38 46 32 40 48

> x.matrix[5,][x.matrix[5,] > 35]
[1] 36 38 40

> x.matrix[,4][x.matrix[,4] > 35]
[1] 40 48

> x.matrix[,1][x.matrix[,1] > 10 & x.matrix[,1] < 30]
[1] 18 26
```

5. Create the following matrix

```
     individual weight age
[1,]          1      4  13
[2,]          2      7  15
[3,]          3      2  12
[4,]          4      9  14
[5,]          5      2  20
```

- Give the columns the names above: use *colnames()*.
- Extract which individuals are older than five years old.
- What is the mean weight of all individuals.
- What is the mean weight of individuals older than five.
- Add a new row - i.e. c(6, 5, 15) - using *rbind()* function.

- Answer

```
> y.matrix = matrix(c(seq(1,5,1),c(4,7,2,9,2,13,15,12,14,20)),
                    nrow=5,ncol=3,byrow=FALSE
                   )

> colnames(y.matrix) = c('individual','weight','age')

> y.matrix
     individual weight age
[1,]          1      4  13
[2,]          2      7  15
[3,]          3      2  12
[4,]          4      9  14
[5,]          5      2  20

> y.matrix[y.matrix[,3] > 5,'individual']
[1] 1 2 3 4 5

or

> y.matrix[y.matrix[,3] > 5,1]
[1] 1 2 3 4 5

> mean(y.matrix[y.matrix[,1],2])
[1] 4.8

> mean(y.matrix[y.matrix[,3] > 5,2])
[1] 4.8

> y.matrix = rbind(y.matrix, c(6,5,15))

> y.matrix
     individual weight age
[1,]          1      4  13
[2,]          2      7  15
[3,]          3      2  12
[4,]          4      9  14
[5,]          5      2  20
[6,]          6      5  15
```

8.3 Operators as functions

Operators can be used like functions. Place the values to supply to the operator function and the result will return as expected.

```
> '+'(1,2)
[1] 3

> '='(x,2)
> x
[1] 2

> '<-'(y,3)

> y
[1] 3
```

Even arguments can be used as functions.

```
> '^'(y,2)
[1] 9
```

9. Output to standard out

9.1 print() and cat()

print(): prints its argument.

cat(): Concatenate and print. This function outputs the objects, concatenating the representations.

cat() performs much less conversion than *print()*.

```
> string = 'My string'

> numbers = 1:12

> t_date = date()

> chain = c(string, numbers, t_date)

> cat(chain)
My string 1 2 3 4 5 6 7 8 9 10 11 12 Tue Sep 18 13:00:07 2018

> print(chain)
 [1] "My string"               "1"
 [3] "2"                       "3"
 [5] "4"                       "5"
 [7] "6"                       "7"
 [9] "8"                       "9"
[11] "10"                      "11"
[13] "12"                      "Tue Sep 18 13:00:07 2018"

> cat(string, numbers, t_date)
My string 1 2 3 4 5 6 7 8 9 10 11 12 Tue Sep 18 13:00:07 2018

> print(string, numbers, t_date)
Error in print.default(string, numbers, t_date) :
  invalid 'quote' argument

> print(c(string, numbers, t_date))
 [1] "My string"               "1"
 [3] "2"                       "3"
 [5] "4"                       "5"
 [7] "6"                       "7"
 [9] "8"                       "9"
[11] "10"                      "11"
[13] "12"                      "Tue Sep 18 13:00:07 2018"
```

9.2 sprintf()

A wrapper for the C function *sprintf*, that returns a character vector containing a formatted combination of text and variable values. The format string is a character string, beginning and ending in its initial shift state, if any. The format string is composed of zero or more directives: ordinary characters (not %), which are copied verbatim to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the character %, and ends with a conversion specifier. In between there may be zero or more flags, an optional minimum *field width*, an optional *precision* and an optional *length modifier*.

Notation	Description
%s	a string
%d	an integer
%0xd	an integer padded with x leading zeros
%f	decimal notation with six decimals
%.xf	floating point number with x digits after decimal point
%e	compact scientific notation, e in the exponent
%E	compact scientific notation, E in the exponent
%g	compact decimal or scientific notation (with e)

9.3 paste() and paste0() functions

paste(): Concatenates vectors after converting to character with a single space as a separator.

paste0(): Concatenates vectors after converting to character with no separator.

- *sep*: The element which separates every term. It should be specified with character string format.
- *collapse*: The element which separates every result. It should be specified with character string format and it is optional.

The difference between *paste()* and *paste0()* is that the argument *sep* by default is " " (paste) and "" in (paste0).

```
> string = 'My string'

> numbers = 1:12

> t_date = date()

> chain = c(string, numbers, t_date)

> p_string = paste(string, numbers, t_date)

> chain
 [1] "My string"               "1"
 [3] "2"                       "3"
 [5] "4"                       "5"
 [7] "6"                       "7"
 [9] "8"                       "9"
[11] "10"                      "11"
[13] "12"                      "Tue Sep 18 13:00:07 2018"

> p_chain
 [1] "My string 1 Tue Sep 18 13:00:07 2018"
 [2] "My string 2 Tue Sep 18 13:00:07 2018"
 [3] "My string 3 Tue Sep 18 13:00:07 2018"
 [4] "My string 4 Tue Sep 18 13:00:07 2018"
 [5] "My string 5 Tue Sep 18 13:00:07 2018"
 [6] "My string 6 Tue Sep 18 13:00:07 2018"
 [7] "My string 7 Tue Sep 18 13:00:07 2018"
 [8] "My string 8 Tue Sep 18 13:00:07 2018"
 [9] "My string 9 Tue Sep 18 13:00:07 2018"
[10] "My string 10 Tue Sep 18 13:00:07 2018"
[11] "My string 11 Tue Sep 18 13:00:07 2018"
[12] "My string 12 Tue Sep 18 13:00:07 2018"
```

9.4 Numbers

Double precision value, in *fixed point* decimal notation.

```
> number = 1234567.894567

> number1 = sprintf("%.2f", number)

> print(number1)
[1] "1234567.89"

> number1 = sprintf("%e", number)

> print(number1)
[1] "1234567.89"

# Exponential output
> number2 = sprintf("%e", number)

> print(number2)
[1] "1.234568e+06"

# Scientific notation
> number3 = sprintf("%a", number)

> print(number3)
[1] "0x1.2d687e50257c9p+20"
```

Note that with integers, if an non-integer is given *R* will give an error. Either give the input as an integer. Optionally print using the floating point number with zero digits after the decimal point to get an integer from a non integer input.

```
# Integer
> number4 = sprintf("%d", number)
Error in sprintf("%d", 1234567.894567) :
  invalid format '%d'; use format %f, %e, %g or %a for numeric objects

> number1 = sprintf("%d", 1234567)
> print(number1)
[1] "1234567"

> number5 = sprintf("%.0f", number)
> print(number5)
[1] "1234568"
```

9.5 Character string

```
> word = "R programming"

> word1 = sprintf("%s is fun.", word)

> print(word1)
[1] "R programming is fun."
```

10. Flow control & Data frames

10.1 if & else conditionals

Used to create flexibility in programming.

Write an *if()* statement that tells if a number (*x*) is positive (the console returns the word *positive*).

```
> x = 5
> y = -4

> if (x > 0) print('positive') else print('negative')
[1] "positive"

> if (y > 0) print('positive') else print('negative')
[1] "negative"

> if (x > 0){
    print('positive')
}else{
    print('negative')
}
[1] "positive"

> if (y > 0){
    print('positive')
}else{
    print('negative')
}
[1] "negative"
```

- *if* & *if else* are not vectorised, they only take single values.

10.2 ifelse conditional

ifelse(*boolean condition,output if TRUE,output if FALSE*)

```
> ifelse((x > 0),'positive','negative')
[1] "positive"

> ifelse((y > 0),'positive','negative')
[1] "negative"
```

ifelse is a vectorised function.

10.2.1 Exercise: ifelse

age = c(3,5,7,4,9,3,5,4,8,6)

- Create a new variable (age.cat) where ages four and below are *0* and above four are *1*.

```
> age = c(3,5,7,4,9,3,5,4,8,6)

> age.cat = ifelse((age < 5),0,1)

> age.cat
 [1] 0 1 1 0 1 0 1 0 1 1
```

- Create a new variable (age.limit) where ages six and above are included in a single age category of *6*.

```
> age.limit = ifelse((age < 6),age,6)

> age.limit
 [1] 3 5 6 4 6 3 5 4 6 6
```

- Create a new variable (age.3cat) where ages are in three categories 1=(1 to 4), 2=(5 to 6), and 3=(7 to 9)

```
> age.3cat = ifelse(
              (age < 5),
              print ('1'),
              ifelse(
                (age > 4 & age < 7),
                print ('2'),
                ifelse(
                  (age > 6),
                  print ('3'),"NA"
                )
              )
            )
[1] "1"
[1] "2"
[1] "3"

> age.3cat
 [1] "1" "2" "3" "1" "3" "1" "2" "1" "3" "2"
```

10.3 Lists

- Lists are the *R* objects which contain elements of different types like – numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using *list()* function.
- Here is an example.

```
> list_data = list("Orange", "Green", c(43,13,61), TRUE, 42.18, 218.2)

> list_data
[[1]]
[1] "Orange"

[[2]]
[1] "Green"

[[3]]
[1] 43 13 61

[[4]]
[1] TRUE

[[5]]
[1] 42.18

[[6]]
[1] 218.2
```

- Accessing elements in the list. Here accessing the forth element in the list.

```
> list_data[4]
[[1]]
[1] TRUE
```

- The *R* List can combine objects of any mode into a single object.

```
> w = list(site=5,species=c(2,3,4,5,6,7,8,9),
           names=c('Captain Kirk','Richard Dawkins')
          )
```

- Use *unclass()* to view the list.

```
> unclass(w)
$site
[1] 5

$species
[1] 2 3 4 5 6 7 8 9

$names
[1] "Captain Kirk"    "Richard Dawkins"
```

- Here are the various tools to extract from the list. These tools all index the second sub-vector of the list.

```
> w$site
[1] 5

> w$species
[1] 2 3 4 5 6 7 8 9

> w[['species']]
[1] 2 3 4 5 6 7 8 9
```

```
> w[[1]]
[1] 5

> w[3]
$names
[1] "Captain Kirk"    "Richard Dawkins"
```

- Adding to the list.

```
> w$shopping.list = c('milk','eggs')

> w
$site
[1] 5

$species
[1] 2 3 4 5 6 7 8 9

$names
[1] "Captain Kirk"    "Richard Dawkins"

$shopping.list
[1] "milk" "eggs"
```

To remove *shopping.list* from our list *w*.

```
> w$shopping.list = NULL

> w
$site
[1] 5

$species
[1] 2 3 4 5 6 7 8 9

$names
[1] "Captain Kirk"    "Richard Dawkins"
```

Most statistical outputs (objects) are contained within an *R* list.

lm(): - is used to fit linear models.
mtcars: - is a stored dataset.

```
> data(mtcars)

> a = lm(mpg~wt, data=mtcars)

> summary(a)

Call:
lm(formula = mpg ~ wt, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max
-4.5432 -2.3647 -0.1252  1.4096  6.8727

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  37.2851     1.8776  19.858  < 2e-16 ***
wt           -5.3445     0.5591  -9.559 1.29e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.046 on 30 degrees of freedom
Multiple R-squared:  0.7528,  Adjusted R-squared:  0.7446
F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```

```
> unclass(a)
$coefficients
(Intercept)          wt
  37.285126   -5.344472

$residuals
          Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4 Drive
         -2.2826106          -0.9197704          -2.0859521           1.2973499
  Hornet Sportabout             Valiant          Duster 360           Merc 240D
         -0.2001440          -0.6932545          -3.9053627           4.1637381
           Merc 230            Merc 280           Merc 280C          Merc 450SE
          2.3499593           0.2998560          -1.1001440           0.8668731
         Merc 450SL         Merc 450SLC  Cadillac Fleetwood Lincoln Continental
         -0.0502472          -1.8830236           1.1733496           2.1032876
  Chrysler Imperial            Fiat 128          Honda Civic      Toyota Corolla
          5.9810744           6.8727113           1.7461954           6.4219792
      Toyota Corona    Dodge Challenger         AMC Javelin          Camaro Z28
         -2.6110037          -2.9725862          -3.7268663          -3.4623553
   Pontiac Firebird            Fiat X1-9       Porsche 914-2        Lotus Europa
          2.4643670           0.3564263           0.1520430           1.2010593
     Ford Pantera L        Ferrari Dino       Maserati Bora          Volvo 142E
         -4.5431513          -2.7809399          -3.2053627          -1.0274952

$effects
 (Intercept)           wt
-113.6497374  -29.1157217   -1.6613339    1.6313943    0.1111305   -0.3840041

  -3.6072442    4.5003125    2.6905817    0.6111305   -0.7888695    1.1143917

   0.2316793   -1.6061571    1.3014525    2.2137818    6.0995633    7.3094734

   2.2421594    6.8956792   -2.2010595   -2.6694078   -3.4150859   -3.1915608

   2.7346556    0.8200064    0.5948771    1.7073457   -4.2045529   -2.4018616

  -2.9072442   -0.6494289

$rank
[1] 2

$fitted.values
          Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4 Drive
          23.282611           21.919770           24.885952           20.102650
  Hornet Sportabout             Valiant          Duster 360           Merc 240D
          18.900144           18.793255           18.205363           20.236262
           Merc 230            Merc 280           Merc 280C          Merc 450SE
          20.450041           18.900144           18.900144           15.533127
         Merc 450SL         Merc 450SLC  Cadillac Fleetwood Lincoln Continental
          17.350247           17.083024            9.226650            8.296712
  Chrysler Imperial            Fiat 128          Honda Civic      Toyota Corolla
           8.718926           25.527289           28.653805           27.478021
      Toyota Corona    Dodge Challenger         AMC Javelin          Camaro Z28
          24.111004           18.472586           18.926866           16.762355
   Pontiac Firebird            Fiat X1-9       Porsche 914-2        Lotus Europa
          16.735633           26.943574           25.847957           29.198941
     Ford Pantera L        Ferrari Dino       Maserati Bora          Volvo 142E
          20.343151           22.480940           18.205363           22.427495

$assign
[1] 0 1

$qr
$qr
                  (Intercept)           wt
Mazda RX4           -5.6568542 -18.199514334
Mazda RX4 Wag        0.1767767   5.447820482
Datsun 710           0.1767767   0.148230003
```

```
Hornet 4 Drive        0.1767767  -0.016055881
Hornet Sportabout     0.1767767  -0.057356801
Valiant               0.1767767  -0.061027994
Duster 360            0.1767767  -0.081219555
Merc 240D             0.1767767  -0.011466889
Merc 230              0.1767767  -0.004124504
Merc 280              0.1767767  -0.057356801
Merc 280C             0.1767767  -0.057356801
Merc 450SE            0.1767767  -0.172999378
Merc 450SL            0.1767767  -0.110589098
Merc 450SLC           0.1767767  -0.119767081
Cadillac Fleetwood    0.1767767  -0.389599760
Lincoln Continental   0.1767767  -0.421539139
Chrysler Imperial     0.1767767  -0.407037927
Fiat 128              0.1767767   0.170257160
Honda Civic           0.1767767   0.277639553
Toyota Corolla        0.1767767   0.237256431
Toyota Corona         0.1767767   0.121613854
Dodge Challenger      0.1767767  -0.072041573
AMC Javelin           0.1767767  -0.056439003
Camaro Z28            0.1767767  -0.130780659
Pontiac Firebird      0.1767767  -0.131698458
Fiat X1-9             0.1767767   0.218900467
Porsche 914-2         0.1767767   0.181270739
Lotus Europa          0.1767767   0.296362637
Ford Pantera L        0.1767767  -0.007795696
Ferrari Dino          0.1767767   0.065628162
Maserati Bora         0.1767767  -0.081219555
Volvo 142E            0.1767767   0.063792566
attr(,"assign")
[1] 0 1

$qraux
[1] 1.176777 1.046354

$pivot
[1] 1 2

$tol
[1] 1e-07

$rank
[1] 2

attr(,"class")
[1] "qr"

$df.residual
[1] 30

$xlevels
named list()

$call
lm(formula = mpg ~ wt, data = mtcars)

$terms
mpg ~ wt
attr(,"variables")
list(mpg, wt)
attr(,"factors")
    wt
mpg  0
wt   1
attr(,"term.labels")
[1] "wt"
attr(,"order")
```

```
[1] 1
attr(,"intercept")
[1] 1
attr(,"response")
[1] 1
attr(,".Environment")
<environment: R_GlobalEnv>
attr(,"predvars")
list(mpg, wt)
attr(,"dataClasses")
      mpg        wt
"numeric" "numeric"

$model
                     mpg    wt
Mazda RX4           21.0 2.620
Mazda RX4 Wag       21.0 2.875
Datsun 710          22.8 2.320
Hornet 4 Drive      21.4 3.215
Hornet Sportabout   18.7 3.440
Valiant             18.1 3.460
Duster 360          14.3 3.570
Merc 240D           24.4 3.190
Merc 230            22.8 3.150
Merc 280            19.2 3.440
Merc 280C           17.8 3.440
Merc 450SE          16.4 4.070
Merc 450SL          17.3 3.730
Merc 450SLC         15.2 3.780
Cadillac Fleetwood  10.4 5.250
Lincoln Continental 10.4 5.424
Chrysler Imperial   14.7 5.345
Fiat 128            32.4 2.200
Honda Civic         30.4 1.615
Toyota Corolla      33.9 1.835
Toyota Corona       21.5 2.465
Dodge Challenger    15.5 3.520
AMC Javelin         15.2 3.435
Camaro Z28          13.3 3.840
Pontiac Firebird    19.2 3.845
Fiat X1-9           27.3 1.935
Porsche 914-2       26.0 2.140
Lotus Europa        30.4 1.513
Ford Pantera L      15.8 3.170
Ferrari Dino        19.7 2.770
Maserati Bora       15.0 3.570
Volvo 142E          21.4 2.780
```

10.4 Table

- Tables return the number of each element in a vector. The vector contains one *a*, three *b* and one *c*.

```
> vector = c("a", "a", "b", "b", "b", "c")

> table(vector)
vector
a b c
2 3 1

> z = table(vector)

> z[2]
b
3
```

10.5 for() and while()

for(): is used to repeat a set of instructions, and it is used when you know in advance the values that the loop variable will have each time it goes through the loop.

while(): is be used to repeat a set of instructions, and it is often used when you do not know in advance how often the instructions will be executed.

```
> for (element in seq(0:10)){
       print(element);
 }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
[1] 11

> numbers = c(1,2,4,8,16,32,64)

> for (loop in numbers){
      cat("Loop number: ",loop,"\n");
 }
Loop number:  1
Loop number:  2
Loop number:  4
Loop number:  8
Loop number:  16
Loop number:  32
Loop number:  64

> cars = list('Ford', 'Saab', 'Toyota', 'Nissan', 'Volvo', 'Renault')

> position = 1

> for (loop in cars){
    cat("Car number: ",position, ':', loop,"\n");
    position = position + 1;
 }
Car number:  1 : Ford
Car number:  2 : Saab
Car number:  3 : Toyota
```

```
Car number:  4 : Nissan
Car number:  5 : Volvo
Car number:  6 : Renault

> position = 1

> while(position < 5){
     print (cars[position]);
     position = position + 1;
 }
[[1]]
[1] "Ford"

[[1]]
[1] "Saab"

[[1]]
[1] "Toyota"

[[1]]
[1] "Nissan"
```

10.6 Data-frame

A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column. A spreadsheet if you will. There MUST be the same number of values in each row and column. Cells cannot be empty and must al least have a logical constant which contains the missing value indicator *NA*.

Following are the characteristics of a data frame.

- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain same number of data items.

Here is an example.

```
> employee.data = data.frame(employee_id = c (1:5),
    employee_name = c("Áine","Dónal","Siobhán","Sinéad","Donnacha"),
    salary = c(623.3,515.2,611.0,729.0,843.25),
    start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
    "2015-03-27")),
    stringsAsFactors = FALSE
   )
```

- Print the data frame.

```
  > employee.data
    employee_id employee_name salary start_date
  1           1          Áine 623.30 2012-01-01
  2           2         Dónal 515.20 2013-09-23
  3           3       Siobhán 611.00 2014-11-15
  4           4        Sinéad 729.00 2014-05-11
  5           5      Donnacha 843.25 2015-03-27
```

- Access elements from the frame.

```
> data.frame(employee.data$employee_name,employee.data$start_date)
  employee.data.employee_name employee.data.start_date
1                       Áine                2012-01-01
2                      Dónal                2013-09-23
3                    Siobhán                2014-11-15
4                     Sinéad                2014-05-11
5                   Donnacha                2015-03-27

> employee.data[1:2]
  employee_id employee_name
1           1          Áine
2           2         Dónal
3           3       Siobhán
4           4        Sinéad
5           5      Donnacha
```

- Extract first two rows.

```
> employee.data[1:2,]
  employee_id employee_name salary start_date
1           1          Áine  623.3 2012-01-01
2           2         Dónal  515.2 2013-09-23
```

- Consider the following *R* dataframe.

```
> individual = c("a1", "a2", "a3", "a4")

> age = c(15,13,16,12)

> weight.class = c("high", "high", "low", "high")

> df = data.frame(individual, age, weight.class)

> df
  individual age weight.class
1         a1  15         high
2         a2  13         high
3         a3  16          low
4         a4  12         high
```

- Use *class()* or *str()* to look at how each variable is classified.

```
> class(individual)
[1] "character"

> class(age)
[1] "numeric"

> class(weight.class)
[1] "character"

> class(df)
[1] "data.frame"

> str(individual)
 chr [1:4] "a1" "a2" "a3" "a4"

> str(age)
 num [1:4] 15 13 16 12

> str(weight.class)
 chr [1:4] "high" "high" "low" "high"
```

```
> str(df)
'data.frame': 4 obs. of  3 variables:
 $ individual  : Factor w/ 4 levels "a1","a2","a3",..: 1 2 3 4
 $ age         : num  15 13 16 12
 $ weight.class: Factor w/ 2 levels "high","low": 1 1 2 1
```

10.7 Indexing Data-frames

read.csv(): Imports data from a Comma Separated File (CSV) file.

head() and **tail():** Return the first or last parts of a vector, matrix, table, data frame or function.

For demonstration purposes, import the *bird_egg.csv* file.

```
> bird_egg = read.csv('Datasets/bird_egg.csv', header=TRUE)

> head(bird_egg)
  individual year clutch age eggs dist_food fail_fledge
1      rm/bg  101      3   2    4       149           1
2      wm/rb   97      3   1    3        63           0
3     rb/bkm  107      3   2    4        NA           0
4    bbk/bkm  108      3   7    3        NA           0
5     wbk/ym  106      3   2    3        NA           1
6       o/ym  103      3   2    4       164           0

> tail(bird_egg)
    individual year clutch age eggs dist_food fail_fledge
825      rr/jm  108      1   2    5        NA           1
826      vm/bv  109      1   1    4        NA           1
827      m/wy2  108      1   3    5        NA           1
828       hm/y  109      1   2    4        NA           1
829        /hm  108      1   1    4        NA           0
830      wm/yr  106      1   4    4        NA           1
```

Reviewing the column for year is (Note: *head()* function used to limit the output to a useable output):

```
> head(bird_egg$year)
[1] 101  97 107 108 106 103

> head(bird_egg[,2])
[1] 101  97 107 108 106 103
```

The first five rows of age is either of these (Note: *head()* function used to limit the output to a useable output):

```
> bird_egg$age[1:5]
[1] 2 1 2 7 2

> bird_egg[1:5,4]
[1] 2 1 2 7 2
```

10.8 Add a new column

Add a column of squared values (e.g. age two) to bird_egg?. It is a simple matter of naming it and assign values to it.

```
> bird_egg$age.square = bird_egg$age^2

> head(bird_egg)
  individual year clutch age eggs dist_food fail_fledge age.square
1      rm/bg  101      3   2    4       149           1          4
2      wm/rb   97      3   1    3        63           0          1
3     rb/bkm  107      3   2    4        NA           0          4
4    bbk/bkm  108      3   7    3        NA           0         49
5     wbk/ym  106      3   2    3        NA           1          4
6       o/ym  103      3   2    4       164           0          4
```

It is also possible to add to the data frame based on an ifelse() decision rule.

This example:

- Adds new column to the *bird_egg* data.frame and calls it *egg.factor*.
- Creates a 2-level categorical variable for number of eggs.

```
> bird_egg$eggs.factor = ifelse(bird_egg$eggs > 3, 'few', 'many')

> head(bird_egg)
individual year clutch age eggs dist_food fail_fledge age.square eggs.factor
1      rm/bg  101      3   2    4       149           1          4         few
2      wm/rb   97      3   1    3        63           0          1        many
3     rb/bkm  107      3   2    4        NA           0          4         few
4    bbk/bkm  108      3   7    3        NA           0         49        many
5     wbk/ym  106      3   2    3        NA           1          4        many
6       o/ym  103      3   2    4       164           0          4         few
```

10.9 Single or double brackets for indexing?

Single and double square brackets. As is demonstrated below the single brackets extract a list whereas double brackets extract in a numeric vector style format.

```
> a = seq(1, 10, length.out = 10)
> b = seq(1, 10, length.out = 10)
> c = seq(1.75, 3.2, length.out = 10)
> d = seq(0.43, 1.6, length.out = 10)

> a = data.frame(w = a, x = b, y = c, z = d)

> m = a[1]
> n = a[[2]]
> o = a[3]
> p = a[[4]]

> class(m)
[1] "data.frame"

> class(n)
[1] "numeric"

> class(o)
[1] "data.frame"

> class(p)
[1] "numeric"
```

```
> typeof(m)
[1] "list"

> typeof(n)
[1] "double"

> typeof(o)
[1] "list"

> typeof(p)
[1] "double"

> m
    w
1   1
2   2
3   3
4   4
5   5
6   6
7   7
8   8
9   9
10 10

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

> o
          y
1  1.750000
2  1.911111
3  2.072222
4  2.233333
5  2.394444
6  2.555556
7  2.716667
8  2.877778
9  3.038889
10 3.200000

> p
 [1] 0.43 0.56 0.69 0.82 0.95 1.08 1.21 1.34 1.47 1.60
```

10.10 Exercise: Data frame 1

Create a new third variable in the data frame (called *age.cat*) which is a categorical variable based on age where all individuals four or less are *young* and those greater than four are considered *old*.

- Answer:

```
> individual = 1:10
> age = c(3,5,7,4,9,3,5,4,8,6)
> age.df = data.frame (individual, age)

> age.df$age.cat = ifelse(age.df$age < 5, 'young', 'old')

> age.df
   individual age age.cat
1           1   3   young
2           2   5     old
3           3   7     old
4           4   4   young
5           5   9     old
6           6   3   young
7           7   5     old
8           8   4   young
9           9   8     old
10         10   6     old
```

10.11 Changing names within the data

It is often necessary to adjust the names within the data. Follow the example below.

```
> names(age.df)
[1] "Individual" "age"  "age.cat"

> names(age.df)[3] = "age2"

> age.df
   Individual age  age2
1           1   3 young
2           2   5   old
3           3   7   old
4           4   4 young
5           5   9   old
6           6   3 young
7           7   5   old
8           8   4 young
9           9   8   old
10         10   6   old
```

10.12 Read files into R

***r*ead.csv()** and **read.csv2():**

- *read.csv* and *read.csv2* are identical to r*ead.table* except for the defaults. They are intended for reading *comma separated value* files (*.csv*) or (*read.csv2*) the variant used in countries that use a comma as decimal point and a semicolon as field separator.

read.table():

- Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

read.delim(): - This is a wrapper function for *read.table()* with default argument values that are convenient when reading in tab-separated data.

10.13 English / European

Many European computer settings are different to the original *.csv* delimited file settings because the comma is used to denote fractions of numbers (decimals).

- 3.14 (English setting)
- 3,14 (European setting)

Thus European computers use semi-colons in their *comma separated* files and commas for decimals. This causes great confusion to *R* if import it the wrong way Open your csv file in a text editor and check how it looks.

- Use *read.csv()* for *English* settings
- Use *read.csv2()* for *European* settings

10.14 Set a working directory

Set a working directory or set file pathway to read in data files.

- **setwd():** - Is used to set the working directory to *dir*.
- **get.wd():** - Check what the working directory is set to.
- **file.choose():** - Choose a File Interactively.

```
> setwd('~/course_datasets')

> ind = read.csv('individual.csv')

> head(ind)
  individual trait
1          a     3
2          a     4
3          g     2
4          g     6
5          g     5
6          g     4
```

10.15 Checks after importing Data-frame

The first steps after importing data.frame are:

1. Check it looks like it should - use *head()* & *tail()*.
2. Use *summary()* to get an overview of the data.
3. Use *str()* to check the structure of each variable.
4. Fix variable classes if necessary with *as.factor()* or *as.character()*, *as.numeric()* or *as.Date()* functions.
5. Get the names of each variable using *names()* (tip: paste names in to your code for future reference).
6. Create the datasets you want to use for analysis by using *data.frame* indexing & filtering.

10.16 Removing missing values from a data set

Complete cases is a logical (boolean) function that returns *TRUE* for each observation (vectors) or row (data frame) that is complete (i.e. has no missing value / NA) for a data.frame called *data*.

```
> complete.data = data[complete.cases(data),]
```

10.16.1 Example:

Use the dataset *bird_egg.csv*.

```
> df = read.csv('bird_egg.csv')

> head(df)
  individual year clutch age eggs dist_food fail_fledge
1      rm/bg  101      3   2    4       149           1
2      wm/rb   97      3   1    3        63           0
3     rb/bkm  107      3   2    4        NA           0
4    bbk/bkm  108      3   7    3        NA           0
5     wbk/ym  106      3   2    3        NA           1
6       o/ym  103      3   2    4       164           0

> summary(df)
   individual       year           clutch           age              eggs
 rm/bg  : 21   Min.   : 97.0   Min.   :1.000   Min.   : 1.000   Min.   :1.000
 bm/bw  : 20   1st Qu.:102.0   1st Qu.:1.000   1st Qu.: 1.000   1st Qu.:4.000
 m/ro   : 19   Median :105.0   Median :1.000   Median : 2.000   Median :4.000
 bb/m   : 17   Mean   :104.7   Mean   :1.396   Mean   : 2.667   Mean   :4.032
 bb/rm  : 16   3rd Qu.:107.0   3rd Qu.:2.000   3rd Qu.: 4.000   3rd Qu.:5.000
 g/m    : 15   Max.   :109.0   Max.   :3.000   Max.   :10.000   Max.   :6.000
 (Other):722                                   NA's   :1        NA's   :5
   dist_food       fail_fledge
 Min.   :  8.00   Min.   :0.0000
 1st Qu.: 10.00   1st Qu.:0.0000
 Median : 70.00   Median :1.0000
 Mean   : 90.69   Mean   :0.6679
 3rd Qu.:135.00   3rd Qu.:1.0000
 Max.   :434.00   Max.   :1.0000
 NA's   :481      NA's   :2

> str(df)
'data.frame': 830 obs. of  7 variables:
 $ individual : Factor w/ 252 levels "","/gm","/hm",..: 162 214 152 11 204 130 234
238 55 214 ...
 $ year       : int  101 97 107 108 106 103 97 97 100 98 ...
 $ clutch     : int  3 3 3 3 3 3 3 3 3 3 ...
 $ age        : int  2 1 2 7 2 2 4 1 2 2 ...
 $ eggs       : int  4 3 4 3 3 4 3 3 2 3 ...
 $ dist_food  : int  149 63 NA NA NA 164 18 191 112 12 ...
 $ fail_fledge: int  1 0 0 0 1 0 0 0 0 0 ...

> df$year = as.factor(df$year)

> df$fail_fledge = as.factor(df$fail_fledge)

> df$clutch = as.factor(df$clutch)

> str(df)
'data.frame': 830 obs. of  7 variables:
 $ individual : Factor w/ 252 levels "","/gm","/hm",..: 162 214 152 11 204 130 234
238 55 214 ...
 $ year       : Factor w/ 13 levels "97","98","99",..: 5 1 11 12 10 7 1 1 4 2 ...
 $ clutch     : Factor w/ 3 levels "1","2","3": 3 3 3 3 3 3 3 3 3 3 ...
 $ age        : int  2 1 2 7 2 2 4 1 2 2 ...
 $ eggs       : int  4 3 4 3 3 4 3 3 2 3 ...
 $ dist_food  : int  149 63 NA NA NA 164 18 191 112 12 ...
 $ fail_fledge: Factor w/ 2 levels "0","1": 2 1 1 1 2 1 1 1 1 1 ...

> names(df)
[1] "individual"  "year"        "clutch"      "age"         "eggs"
[6] "dist_food"   "fail_fledge"
```

10.17 Data.frame object classes

Integer	as.integer()
Numeric	as.numeric()
Character	as.character()
Factor	as.factor()
Date	as.Date()
Logical	as.logical()

Examples to change object class.

```
> df$clutch = as.factor(df$clutch)

> df$individual = as.character(df$individual)
```

as.matrix() and *as.data.frame()* can be used for 2-dimensional objects.

10.18 Saving tables & Data Frames

write.table(): - prints its required argument 'x' (after converting it to a data frame if it is not one nor a matrix) to a file or connection.
write.csv(): - Does the same as *write.table()* except in comma delimited format.

These will save the data-frame or matrix to a file in the current working directory (use *get.wd()* to check where that is and *set.wd()* to change it.)

```
> write.csv(df.egg, "bird_egg2.csv")
```

10.19 subset() function

It is regular that a subset of a frame is necessary. The *subset()* function handles this event.

10.19.1 Columns subset

```
> names(df.egg)
[1] "individual"  "year"        "clutch"      "age"         "eggs"
[6] "dist_food"   "fail_fledge"

> df.sub1 = subset(df.egg, select = c(individual, clutch, age, eggs))

> head(df.sub1)
  individual clutch age eggs
1      rm/bg      3   2    4
2      wm/rb      3   1    3
3     rb/bkm      3   2    4
4    bbk/bkm      3   7    3
5     wbk/ym      3   2    3
6       o/ym      3   2    4
```

Obviously the same could be achieved by indexing.

```
> df.sub2 = df.egg[, c(1,3,4,5)]
```

```
> head(df.sub2)
  character factors age eggs
1     rm/bg       3   2    4
2     wm/rb       3   1    3
3    rb/bkm       3   2    4
4   bbk/bkm       3   7    3
5    wbk/ym       3   2    3
6      o/ym       3   2    4
```

10.20 Date and time

Date and time for timestamps or dates and time within datasets can be useful. First looking at getting the time and date from the system with the *date()* function. The example demonstrates getting the date and time plus formatting them using the format types documented in the table below.

```
> t = Sys.time()

> print(t)
[1] "2018-09-28 10:45:41 EAT"

> str(t)
 POSIXct[1:1], format: "2018-09-28 10:45:41

> t.short = format( x = t, format = '%d %b %Y')

> print(t.short)
[1] "28 Sep 2018"

> str(t.short)
 chr "28 Sep 2018"
```

Strings of data that are in *character* or other formats can be converted to a *date* format. First ensure the data is in character format, then apply the *as.Date()* function to change to a date format.

```
> c.date = 'Fri 02 Sep 2018 - 11:02:12'

> c.date = as.character('Fri 02 Sep 2018 - 11:02:12')   # Must be 'chr' format

> str(c.date)
 chr "Fri 02 Sep 2018 - 11:02:12"

> c.date = as.Date(t, format = '%a %d %b %Y - %X')

> print(c.date)
[1] "2018-09-28"

> str(c.date)
 Date[1:1], format: "2018-09-28"

> c.date.short = format( x = c.date, format = '%d %b %Y')

> print(c.date.short)
[1] "28 Sep 2018"

> str(c.date.short)
 chr "28 Sep 2018"
```

For more granularity with time there is a specific *time()* function that creates the vector of times at which a time series was sampled.

Format types

%a	Locale's abbreviated weekday name.
%A	Locale's full weekday name.
%b	Locale's abbreviated month name.
%B	Locale's full month name.
%c	Locale's appropriate date and time representation.
%C	Century (a year divided by 100 and truncated to an integer) as a decimal number [00,99].
%d	Day of the month as a decimal number [01,31].
%D	Date in the format mm/dd/yy.
%e	Day of the month as a decimal number [1,31] in a two-digit field with leading space character fill.
%h	A synonym for %b.
%H	Hour (24-hour clock) as a decimal number [00,23].
%I	Hour (12-hour clock) as a decimal number [01,12].
%j	Day of the year as a decimal number [001,366].
%m	Month as a decimal number [01,12].
%M	inute as a decimal number [00,59].
%n	A <newline>.
%p	Locale's equivalent of either AM or PM.
%r	12-hour clock time [01,12] using the AM/PM notation; in the POSIX locale, this shall be equivalent to %I : %M : %S %p.
%S	Seconds as a decimal number [00,60].
%t	A <tab>.
%T	24-hour clock time [00,23] in the format HH:MM:SS.
%u	Weekday as a decimal number [1,7] (1=Monday).
%U	Week of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday shall be considered to be in week 0.
%V	Week of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing January 1 has four or more days in the new year, then it shall be considered week 1; otherwise, it shall be the last week of the previous year, and the next week shall be week 1.
%w	Weekday as a decimal number [0,6] (0=Sunday).
%W	Week of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday shall be considered to be in week 0.
%x	Locale's appropriate date representation.
%X	Locale's appropriate time representation.
%y	Year within century [00,99].
%Y	Year with century as a decimal number.
%Z	Timezone name, or no characters if no timezone is determinable.

10.21 lapply()

Another function from the *apply()* functions.

The list apply *lapply()* function operates like apply *()* with similar output. The *lapply()* function can operate on other objects like dataframes and lists. It returns a list of the same length as *X*, each element of which is the result of applying *FUN* to the corresponding element of *X*.

- **lapply():** returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.
- **apply(*X, FUN, ...*)**
  - *X*: an array, including a matrix.
  - *FUN*: the function.

This example shows the *mean()* function applied to the elements of the vector at position three of the list.

```
> l = list("Orange", "Green", c(43,13,61), TRUE, 42.18, 218.2)
(l,"[", , 3)

> str(l)
List of 6
 $ : chr "Orange"
 $ : chr "Green"
 $ : num [1:3] 43 13 61
 $ : logi TRUE
 $ : num 42.2
 $ : num 218

> lapply(l[3],mean)
[[1]]
[1] 39

> df = data.frame(employee_id = c (1:5),
       employee_name = c("Áine","Dónal","Siobhán","Sinéad","Donnacha"),
       salary = c(623.3,515.2,611.0,729.0,843.25),
       start_date  =  as.Date(c("2012-01-01",  "2013-09-23",  "2014-11-15",  "2014-05-
       11", "2015-03-27")),
       stringsAsFactors = FALSE
      )

> df
  employee_id employee_name salary start_date
1           1          Áine 623.30 2012-01-01
2           2         Dónal 515.20 2013-09-23
3           3       Siobhán 611.00 2014-11-15
4           4        Sinéad 729.00 2014-05-11
5           5      Donnacha 843.25 2015-03-27

> lapply(df[3],sum)
$salary
[1] 3321.75
```

10.22 User-defined Functions

One of the great strengths of any programming language is the ability to add functions, *R* is no different. A function should be considered if there is any block of code that is being repeated in a script. Instead put the block of code in a function, feed the function values and return results.

```
# Defining function above main block of code
new_function_name = function(arg1, arg2, ... ){
    statements
    return(object)
}

# Call function from main block of code
new_function_name(arg1, arg2, ... )
```

The following script demonstrates how a user-defined function works. Create the *my_function_demo.R* file, make it executable and run it. The steps breakdown:

1. The existing set of objects in *R* are cleared.
2. Five vector objects are created and the last four are added to a dataframe.
3. The user-defined function is defined.
4. The main program follows:
    - A loop of four sends the *v* vector plus the each vector from the dataframe in turn as a list to the user-defined function.
    - The function processes and returns the linear model to where it was called from.
    - This response is assigned to the object *model<loop #>*. In this way four models are defined.
    - The *model<loop #>* names are extracted from the list of objects.
    - A loop through these models and their summary is output to standard out.

```
$ cat << EOM >> my_function_demo.R
#!/usr/bin/Rscript

# Clear objects from R
rm(list = ls())

# Define objects
v = 1:8
v1 = c(51,19,43,74,45,26,83,42)
v2 = c(101,111,112,123,141,152,193,141)
v3 = c(214,233,234,226,237,248,269,276)
v4 = c(322,354,385,377,381,314,425,416)
df = data.frame(v1,v2,v3,v4)

# Function 'realmod'
realmod = function(arg1, arg2){
    x = as.numeric(unlist(arg1));
    y = as.numeric(unlist(arg2));
    return(lm(y ~ x))
}

# Main program (calling the realmod function)
for (x in 1:4){
    assign(paste0('model',x), realmod(v,df[x]));
}
```

```
# Get list objects
obj_list = ls()
mod_list = grep ('model', obj_list)

# Loop through models and output to stdout
for (x in mod_list){
    cat(sprintf('%s',obj_list[x]),'\n');
    cat(strrep("=",6), "\n");
    print(summary(get(obj_list[x])))
}

# End script
quit(status = 0)

EOM

$ chmod +x my_function_demo.R

$ ./my_function_demo.R
model1
======

Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-25.036 -15.839  -2.821  14.670  29.857

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   38.393     17.764   2.161   0.0739 .
x              2.107      3.518   0.599   0.5711
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 22.8 on 6 degrees of freedom
Multiple R-squared:  0.05643, Adjusted R-squared:  -0.1008
F-statistic: 0.3588 on 1 and 6 DF,  p-value: 0.5711

model2
======

Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-27.750  -6.607   1.321   2.107  34.107

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   89.893     14.384   6.250 0.000778 ***
x              9.857      2.848   3.461 0.013459 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 18.46 on 6 degrees of freedom
Multiple R-squared:  0.6662,  Adjusted R-squared:  0.6106
F-statistic: 11.98 on 1 and 6 DF,  p-value: 0.01346

model3
======

Call:
lm(formula = y ~ x)
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-12.155  -6.801   1.726   6.319  10.726

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  206.393      6.989  29.531    1e-07 ***
x              7.940      1.384   5.737  0.00122 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8.97 on 6 degrees of freedom
Multiple R-squared:  0.8458,  Adjusted R-squared:  0.8201
F-statistic: 32.91 on 1 and 6 DF,  p-value: 0.001218

model4
======

Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-72.107  -0.714   8.107  14.964  29.321

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  328.679     27.290  12.044 1.99e-05 ***
x              9.571      5.404   1.771    0.127
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 35.02 on 6 degrees of freedom
Multiple R-squared:  0.3433,  Adjusted R-squared:  0.2339
F-statistic: 3.137 on 1 and 6 DF,  p-value: 0.1269
```

10.23 Recap exercises

10.23.1 Exercise: Data-frames 2

1. Import the *bird_egg.csv* data and call the data frame *df.egg*.
2. Explore the data frame and check structure - change *year*, *fail_fledge* and *clutch* to *factors*, & *individual* to *character*.
3. Create new column in *df.egg* called *constant* consisting of a column of *1*s.
4. Create a new data frame (by indexing) consisting of *individual*, *clutch*, *age* & *egg* and call it *df.sub1*.
5. Create a new data.frame (called *df.sub2*) which consists of all data from first clutches for birds less than three years old.
6. Create a new data.frame (called df.sub3) which contains all data for which there are no missing values...
    - HINT: complete.cases().
7. Calculate the mean distance to food for failed versus fledged nests. Do this for all data.frames (df.sub1, 2 & 3)....
    - HINT: check *help(mean)* if having problems.
8. Create a new column (*dist.cat*) that is a three-level distance category based on the *dist_food* column. Where <100 is *near*, 100-200 is *mid*, and >200 is *far*.
    - HINT: *ifelse(... , ... , ifelse(... , ... , ...))*.

9. Create *df.sub4* which contains all records without missing values in order of bird *age* (youngest to oldest)
    - HINT: *df.sub3[order(),]*.
10. Rename the *individual* column to *id* & rename the *dist_food* column to *distance.food*.
11. Save the new data.frames as *.csv* files in your working directory folder (e.g. *df.sub1.csv* etc).

Answer:

```
##### // Read in the file //

> setwd('~/datasets/birds_egg')

> df.egg = read.csv('bird_egg.csv')


##### // Explore the data frame //

> summary(df.egg)
   individual       year           clutch           age              eggs
 rm/bg  : 21   Min.   : 97.0   Min.   :1.000   Min.   : 1.000   Min.   :1.000
 bm/bw  : 20   1st Qu.:102.0   1st Qu.:1.000   1st Qu.: 1.000   1st Qu.:4.000
 m/ro   : 19   Median :105.0   Median :1.000   Median : 2.000   Median :4.000
 bb/m   : 17   Mean   :104.7   Mean   :1.396   Mean   : 2.667   Mean   :4.032
 bb/rm  : 16   3rd Qu.:107.0   3rd Qu.:2.000   3rd Qu.: 4.000   3rd Qu.:5.000
 g/m    : 15   Max.   :109.0   Max.   :3.000   Max.   :10.000   Max.   :6.000
 (Other):722                                   NA's   :1        NA's   :5
   dist_food       fail_fledge
 Min.   :  8.00   Min.   :0.0000
 1st Qu.: 10.00   1st Qu.:0.0000
 Median : 70.00   Median :1.0000
 Mean   : 90.69   Mean   :0.6679
 3rd Qu.:135.00   3rd Qu.:1.0000
 Max.   :434.00   Max.   :1.0000
 NA's   :481      NA's   :2

> str(df.egg)
'data.frame': 830 obs. of  7 variables:
 $ individual : Factor w/ 252 levels "","/gm","/hm",..: 162 214 152 11 204 130 234
 238 55 214 ...
 $ year       : int  101 97 107 108 106 103 97 97 100 98 ...
 $ clutch     : int  3 3 3 3 3 3 3 3 3 3 ...
 $ age        : int  2 1 2 7 2 2 4 1 2 2 ...
 $ eggs       : int  4 3 4 3 3 4 3 3 2 3 ...
 $ dist_food  : int  149 63 NA NA NA 164 18 191 112 12 ...
 $ fail_fledge: int  1 0 0 0 1 0 0 0 0 0 ...

> names(df.egg)
[1] "individual"  "year"        "clutch"      "age"         "eggs"
[6] "dist_food"   "fail_fledge"

> head(df.egg)
  individual year clutch age eggs dist_food fail_fledge
1      rm/bg  101      3   2    4       149           1
2      wm/rb   97      3   1    3        63           0
3     rb/bkm  107      3   2    4        NA           0
4    bbk/bkm  108      3   7    3        NA           0
5     wbk/ym  106      3   2    3        NA           1
6       o/ym  103      3   2    4       164           0
```

```
##### // Change year, fail_fledge and clutch to factors //

> str(df.egg)
'data.frame': 830 obs. of  8 variables:
 $ individual : Factor w/ 252 levels "","/gm","/hm",..: 162 214 152 11 204 130 234
 238 55 214 ...
 $ year       : int  101 97 107 108 106 103 97 97 100 98 ...
 $ clutch     : int  3 3 3 3 3 3 3 3 3 3 ...
 $ age        : int  2 1 2 7 2 2 4 1 2 2 ...
 $ eggs       : int  4 3 4 3 3 4 3 3 2 3 ...
 $ dist_food  : int  149 63 NA NA NA 164 18 191 112 12 ...
 $ fail_fledge: int  1 0 0 0 1 0 0 0 0 0 ...

> df.egg[, 'year'] = as.factor(df.egg[, 'year'])

> str(df.egg$year)
 Factor w/ 13 levels "97","98","99",..: 5 1 11 12 10 7 1 1 4 2 ...

> df.egg[, 'fail_fledge'] = as.factor(df.egg[, 'fail_fledge'])

> str(df.egg$fail_fledge)
 Factor w/ 2 levels "0","1": 2 1 1 1 2 1 1 1 1 1 ...


##### // Change individual to character //

> df.egg[, 'individual'] =  as.character(df.egg[, 'individual'])

> str(df.egg$individual)
 chr [1:830] "rm/bg" "wm/rb" "rb/bkm" "bbk/bkm" "wbk/ym" "o/ym" "yg/m" ...


##### // Create new column in df.egg called 'constant' made of 1s //

> df.egg$constant = 1

> head(df.egg)
  individual year clutch age eggs dist_food fail_fledge constant
1      rm/bg  101      3   2    4       149           1        1
2      wm/rb   97      3   1    3        63           0        1
3     rb/bkm  107      3   2    4        NA           0        1
4    bbk/bkm  108      3   7    3        NA           0        1
5     wbk/ym  106      3   2    3        NA           1        1
6       o/ym  103      3   2    4       164           0        1

##### // Create a new data frame df.sub1 with individual, clutch, age & eggs //

> names(df.egg)
[1] "individual"  "year"        "clutch"      "age"         "eggs"
[6] "dist_food"   "fail_fledge" "constant"

> df.sub1 = df.egg[, c(1,3,4,5)]

> head(df.sub1)
  individual clutch age eggs
1      rm/bg      3   2    4
2      wm/rb      3   1    3
3     rb/bkm      3   2    4
4    bbk/bkm      3   7    3
5     wbk/ym      3   2    3
6       o/ym      3   2    4


##### // df.sub2 to consists of all data for birds < 3 //

> df.sub2 = df.egg[df.egg$age < 3,]
```

```
> head(df.sub2)
  individual year clutch age eggs dist_food fail_fledge constant
1      rm/bg  101      3   2    4       149           1        1
2      wm/rb   97      3   1    3        63           0        1
3     rb/bkm  107      3   2    4        NA           0        1
5     wbk/ym  106      3   2    3        NA           1        1
6       o/ym  103      3   2    4       164           0        1
8       ym/b   97      3   1    3       191           0        1

##### // df.sub3 to contain all data with no missing values //

> df.sub3 = df.egg[complete.cases(df.egg),]

> head(df.sub3)
  individual year clutch age eggs dist_food fail_fledge constant
1      rm/bg  101      3   2    4       149           1        1
2      wm/rb   97      3   1    3        63           0        1
6       o/ym  103      3   2    4       164           0        1
7       yg/m   97      3   4    3        18           0        1
8       ym/b   97      3   1    3       191           0        1
9       bw/m  100      3   2    2       112           0        1

##### // Calculate the mean distance to food for failed versus fledged nests //

Note: cannot do df.dub1 as it doesn't have a 'fail_fledge' column.

> mean(df.sub2[df.sub2$fail_fledge == 0 & complete.cases(df.sub2$dist_food),6])
[1] 118.2319

> mean(df.sub2[df.sub2$fail_fledge == 1 & complete.cases(df.sub2$dist_food),6])
[1] 93.60135

> mean(df.sub3[df.sub3$fail_fledge == 0 & complete.cases(df.sub3$dist_food),6])
[1] 96.70642

> mean(df.sub3[df.sub3$fail_fledge == 1 & complete.cases(df.sub3$dist_food),6])
[1] 87.9625

##### // 3-level distance category based on the 'dist_food' column //

> df.egg$dist.cat = ifelse(
                     (df.egg$dist_food < 100),
                     print ('near'),
                     ifelse(
                       (df.egg$dist_food > 100 & df.egg$dist_food < 200),
                       print ('mid'),
                       ifelse(
                         (df.egg$dist_food > 200),
                         print ('far'),
                        "NA"
                        )
                      )
                   )
[1] "near"
[1] "mid"
[1] "far"

> head(df.egg)
  individual year clutch age eggs dist_food fail_fledge constant dist.cat
1      rm/bg  101      3   2    4       149           1        1      mid
2      wm/rb   97      3   1    3        63           0        1     near
3     rb/bkm  107      3   2    4        NA           0        1     <NA>
4    bbk/bkm  108      3   7    3        NA           0        1     <NA>
5     wbk/ym  106      3   2    3        NA           1        1     <NA>
6       o/ym  103      3   2    4       164           0        1      mid
```

```
##### // df.sub4 contains all records without missing values in order of bird age //

> temp = df.egg[complete.cases(df.egg),]

> df.sub4 = temp[order(temp$age),]

> head(df.sub4)
   individual year clutch age eggs dist_food fail_fledge constant dist.cat
2       wm/rb   97      3   1    3        63           0        1     near
8        ym/b   97      3   1    3       191           0        1      mid
15       m/ro   99      3   1    4       127           1        1      mid
27       bg/m  100      2   1    4       135           0        1      mid
28      rm/bg  100      2   1    4       150           0        1      mid
31      wm/rb   97      2   1    4        32           0        1     near

##### // rename 'individual' --> 'id', 'dist_food' --> 'distance.food' //

> names(df.egg)
[1] "individual"  "year"        "clutch"      "age"         "eggs"
[6] "dist_food"   "fail_fledge" "constant"    "dist.cat"

> names(df.egg)[1] = "id"

> names(df.egg)[6] = "distance.food"

> names(df.egg)
[1] "id"            "year"          "clutch"        "age"
[5] "eggs"          "distance.food" "fail_fledge"   "constant"
[9] "dist.cat"



> write.csv(df.sub1, file = "df.sub1.csv")

> write.csv(df.sub2, file = "df.sub2.csv")

> write.csv(df.sub3, file = "df.sub3.csv")

> write.csv(df.sub4, file = "df.sub4.csv")
```

10.23.2 Exercise: Data-frames 3

1. Create a new folder that contains the *owl_data.csv* data file.
2. Set your working directory to the new folder.
3. Import the *owl_data.csv* data and call the data frame owl.
4. Explore the data frame and check structure [e.g. by using summary(), str(), names(), head()].
5. Create a column of 1's and 0's called *sex10* where *male=0* & *female=1* (from the *sex* column).
6. Change the *sex10* column to a factor
   - HINT: *as.factor()*.
7. Subset the data.frame to only include data from broods with five chicks.
8. Add 2 columns to the *owl data.frame*.
   - *food.category*: where food is a 2-category variable *low* (food less than 25) & *high* (food greater than or equal to 25).
   - *begging.3*: where begging is a 3-category variable *1* (between 0-10), *2* (from 10-20) and *3* (above 20).
   - HINT: you can nest *ifelse* functions within *ifelse* functions.
9. Save the new data.frame as a *.csv* file called *owl_2.csv*.

```
##### // Inport the owl.csv file //

> setwd('~/datasets/owl_data')

> owl = read.csv('owl_data.csv')


##### // Explore the data frame //

> summary(owl)
          nest         sex           food          begging          brood       
 Oleyes     : 52   Female:245   Min.   :21.71   Min.   : 0.00   Min.   :1.000  
 Moutet     : 41   Male  :354   1st Qu.:23.11   1st Qu.: 0.00   1st Qu.:4.000  
 Etrabloz   : 34                Median :24.38   Median : 5.00   Median :4.000  
 Yvonnand   : 34                Mean   :24.76   Mean   : 6.72   Mean   :4.392  
 Champmartin: 30                3rd Qu.:26.25   3rd Qu.:11.00   3rd Qu.:5.000  
 Lucens     : 29                Max.   :29.25   Max.   :32.00   Max.   :7.000  
 (Other)    :379                                                               

> str(owl)
'data.frame': 599 obs. of  5 variables:
 $ nest   : Factor w/ 27 levels "AutavauxTV","Bochet",..: 1 1 1 1 1 1 1 1 1 ...
 $ sex    : Factor w/ 2 levels "Female","Male": 2 2 2 2 2 2 1 2 1 ...
 $ food   : num  22.2 22.4 22.5 22.6 22.6 ...
 $ begging: int  4 0 2 2 2 18 4 18 0 ...
 $ brood  : int  5 5 5 5 5 5 5 5 5 ...

> names(owl)
[1] "nest"    "sex"     "food"    "begging" "brood"

> head(owl)
        nest  sex  food begging brood
1 AutavauxTV Male 22.25       4     5
2 AutavauxTV Male 22.38       0     5
3 AutavauxTV Male 22.53       2     5
4 AutavauxTV Male 22.56       2     5
5 AutavauxTV Male 22.61       2     5
6 AutavauxTV Male 22.65       2     5


##### // Create the 'sex10' column //

> owl$sex10 = ifelse(owl$sex == 'Male', 0, 1)

> head(owl)
        nest  sex  food begging brood sex10
1 AutavauxTV Male 22.25       4     5     0
2 AutavauxTV Male 22.38       0     5     0
3 AutavauxTV Male 22.53       2     5     0
4 AutavauxTV Male 22.56       2     5     0
5 AutavauxTV Male 22.61       2     5     0
6 AutavauxTV Male 22.65       2     5     0


##### // Change the 'sex10' column values to factors //

> owl[, 'sex10'] = as.factor(owl[, 'sex10'])

> str(owl$sex10)
 Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 2 1 2 ...


##### // Subset to only include data from broods with 5 chicks //

> owl.5chicks = owl[owl$brood == 5,]
```

```
##### // add 'food.category' and 'begging.3' columns //

> owl$food.category = ifelse(owl$food < 25, 'low', 'high')

> head(owl)
        nest  sex  food begging brood sex10 food.category
1 AutavauxTV Male 22.25       4     5     0           low
2 AutavauxTV Male 22.38       0     5     0           low
3 AutavauxTV Male 22.53       2     5     0           low
4 AutavauxTV Male 22.56       2     5     0           low
5 AutavauxTV Male 22.61       2     5     0           low
6 AutavauxTV Male 22.65       2     5     0           low

> owl$begging.3 = ifelse(
                  (owl$begging < 10),
                  print ('1'),
                  ifelse(
                    (owl$begging > 9 & owl$begging < 20),
                    print ('2'),
                    ifelse(
                      (owl$begging > 19),
                      print ('3'),
                    "NA"
                    )
                  )
               )
[1] "1"
[1] "2"
[1] "3"

> head(owl)
        nest  sex  food begging brood sex10 food.category begging.3
1 AutavauxTV Male 22.25       4     5     0           low         1
2 AutavauxTV Male 22.38       0     5     0           low         1
3 AutavauxTV Male 22.53       2     5     0           low         1
4 AutavauxTV Male 22.56       2     5     0           low         1
5 AutavauxTV Male 22.61       2     5     0           low         1
6 AutavauxTV Male 22.65       2     5     0           low         1

##### // Dave the data //

> write.csv(owl, file = "owl_2.csv")
```

10.23.3 Exercise: Data-frames 4

1. Create a new folder that contains the *individual.csv* data file.
2. Set your working directory to the new folder.
3. Import the *individual.csv* data and call the data frame ind.
4. Explore the data frame and check structure. [e.g. by using *summary()*, *str()*, *names()*, *head()*].
5. Create a new column (*id*) where the individual names from the first column are replaced by numbers starting at one and increasing sequentially. Note that individuals are repeated throughout the dataset so you'll have to make sure each number always matches the same individual.
    - HINT: *str()* & *as.numeric()*.
6. Save the new data.frame as a *.csv* file called *ind_2.csv*.

```
##### // Inport the individual.csv file //

> setwd('~/datasets/individual')

> ind = read.csv('individual.csv')


##### // Explore the data frame and check structure.

> summary(ind)
   individual     trait
 g      :9    Min.   :2.000
 n      :6    1st Qu.:4.000
 ee     :5    Median :6.000
 h      :5    Mean   :5.513
 a      :3    3rd Qu.:7.000
 q      :3    Max.   :9.000
 (Other):8

> str(ind)
'data.frame': 39 obs. of  2 variables:
 $ individual: Factor w/ 10 levels "a","ee","g","h",..: 1 1 3 3 3 3 3 3 3 4 ...
 $ trait     : int  3 4 2 6 5 4 6 7 5 4 ...

> names(ind)
[1] "individual" "trait"

> head(ind)
  individual trait
1          a     3
2          a     4
3          g     2
4          g     6
5          g     5
6          g     4


##### // Create column 'id' //

> ind$id = as.numeric(ind[,1])

> ind
   individual trait id
1           a     3  1
2           a     4  1
3           g     2  3
4           g     6  3
5           g     5  3
6           g     4  3
7           g     6  3
8           g     7  3
9           g     5  3
10          h     4  4
11          n     5  6
12          a     7  1
13          n     2  6
14          n     6  6
15          n     7  6
16          n     2  6
17          n     9  6
18         ee     8  2
19         ee     6  2
20         ee     7  2
21         ee     8  2
22          s     9  8
23          h     6  4
24          h     4  4
```

```
25          g     3  3
26          g     6  3
27         ee     7  2
28          h     8  4
29          h     6  4
30          q     5  7
31          q     4  7
32          q     3  7
33          t     6  9
34          t     7  9
35          y     8 10
36          y     8 10
37          t     5  9
38          m     4  5
39          m     3  5
```

```
##### //Save the new as 'ind_2.csv' //

> write.csv(ind, file = "ind_2.csv")
```

10.23.4 Exercise: Data-frames 5

1. Create a new folder that contains the *RIKZ.csv* data file.
2. Set your working directory to the new folder.
3. Import the data (rikz), explore and check structure.
4. reorder the rows of the entire data.frame by increasing values of NAP
    - HINT: rikz[order()].
5. create a data.frame called *rikz2* that only contains data from beaches one to four for the first week of data collection.
6. what is the mean chalk value for Beach 5 and the mean grain size for beach eight?.
7. save the new data.frame as *.csv* file called *rikz_2.csv*.

```
##### // Inport the RIKZ.csv dataset file //

> setwd('~/datasets/RIKZ')

> rikz = read.csv('RIKZ.csv')

##### // Import the data (rikz), explore and check structure //

> summary(rikz)
     Sample      Richness           Week           angle1           angle2
 Min.   : 1   Min.   : 0.000   Min.   :1.000   Min.   :  6.00   Min.   :21.00
 1st Qu.:12   1st Qu.: 3.000   1st Qu.:2.000   1st Qu.: 22.00   1st Qu.:32.00
 Median :23   Median : 4.000   Median :2.000   Median : 32.00   Median :42.00
 Mean   :23   Mean   : 5.689   Mean   :2.333   Mean   : 50.31   Mean   :57.78
 3rd Qu.:34   3rd Qu.: 8.000   3rd Qu.:3.000   3rd Qu.: 55.00   3rd Qu.:89.00
 Max.   :45   Max.   :22.000   Max.   :4.000   Max.   :312.00   Max.   :96.00
    exposure        salinity     temperature         NAP
 Min.   : 8.00   Min.   :26.4   Min.   :15.80   Min.   :-1.3360
 1st Qu.:10.00   1st Qu.:27.1   1st Qu.:17.50   1st Qu.:-0.3750
 Median :10.00   Median :27.9   Median :18.77   Median : 0.1670
 Mean   :10.22   Mean   :28.1   Mean   :18.77   Mean   : 0.3477
 3rd Qu.:11.00   3rd Qu.:29.4   3rd Qu.:20.00   3rd Qu.: 1.1170
 Max.   :11.00   Max.   :29.9   Max.   :20.80   Max.   : 2.2550
 penetrability     grainsize         humus             chalk
 Min.   :151.8   Min.   :186.0   Min.   :0.00000   Min.   : 0.850
 1st Qu.:237.1   1st Qu.:222.5   1st Qu.:0.00000   1st Qu.: 2.200
 Median :256.1   Median :266.0   Median :0.05000   Median : 4.750
 Mean   :289.4   Mean   :272.5   Mean   :0.05028   Mean   : 7.961
```

```
 3rd Qu.:272.9   3rd Qu.:316.5   3rd Qu.:0.10000   3rd Qu.: 9.150
 Max.   :624.0   Max.   :405.5   Max.   :0.30000   Max.   :45.750
    sorting1          Beach
 Min.   : 53.08   Min.   :1
 1st Qu.: 72.75   1st Qu.:3
 Median : 89.03   Median :5
 Mean   : 97.82   Mean   :5
 3rd Qu.:115.44   3rd Qu.:7
 Max.   :248.60   Max.   :9
```

```
> str(rikz)
'data.frame': 45 obs. of  15 variables:
 $ Sample       : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Richness     : int  11 10 13 11 10 8 9 8 19 17 ...
 $ Week         : int  1 1 1 1 1 1 1 1 1 ...
 $ angle1       : int  32 62 65 55 23 129 126 52 26 143 ...
 $ angle2       : int  96 96 96 96 96 89 89 89 89 89 ...
 $ exposure     : int  10 10 10 10 10 8 8 8 8 8 ...
 $ salinity     : num  29.4 29.4 29.4 29.4 29.4 29.6 29.6 29.6 29.6 29.6 ...
 $ temperature  : num  17.5 17.5 17.5 17.5 17.5 20.8 20.8 20.8 20.8 20.8 ...
 $ NAP          : num  0.045 -1.036 -1.336 0.616 -0.684 ...
 $ penetrability: num  254 227 237 249 252 ...
 $ grainsize    : num  222 200 194 221 202 ...
 $ humus        : num  0.05 0.3 0.1 0.15 0.05 0.1 0.1 0.1 0.15 0 ...
 $ chalk        : num  2.05 2.5 3.45 1.6 2.45 2.5 1.85 1.7 2.3 2.6 ...
 $ sorting1     : num  69.8 59 59.2 67.8 57.8 ...
 $ Beach        : int  1 1 1 1 1 2 2 2 2 ...
```

```
> names(rikz)
 [1] "Sample"        "Richness"      "Week"          "angle1"
 [5] "angle2"        "exposure"      "salinity"      "temperature"
 [9] "NAP"           "penetrability" "grainsize"     "humus"
[13] "chalk"         "sorting1"      "Beach"
```

```
> head(rikz)
  Sample Richness Week angle1 angle2 exposure salinity temperature    NAP
1      1       11    1     32     96       10     29.4        17.5  0.045
2      2       10    1     62     96       10     29.4        17.5 -1.036
3      3       13    1     65     96       10     29.4        17.5 -1.336
4      4       11    1     55     96       10     29.4        17.5  0.616
5      5       10    1     23     96       10     29.4        17.5 -0.684
6      6        8    1    129     89        8     29.6        20.8  1.190
  penetrability grainsize humus chalk sorting1 Beach
1         253.9     222.5  0.05  2.05   69.830     1
2         226.9     200.0  0.30  2.50   59.000     1
3         237.1     194.5  0.10  3.45   59.220     1
4         248.6     221.0  0.15  1.60   67.750     1
5         251.9     202.0  0.05  2.45   57.760     1
6         250.1     192.5  0.10  2.50   53.075     2
```

// Reorder the rows of the entire data.frame by increasing values of NAP. HINT: rikz[order()] //

```
> rikz = rikz[order(rikz$NAP),]
```

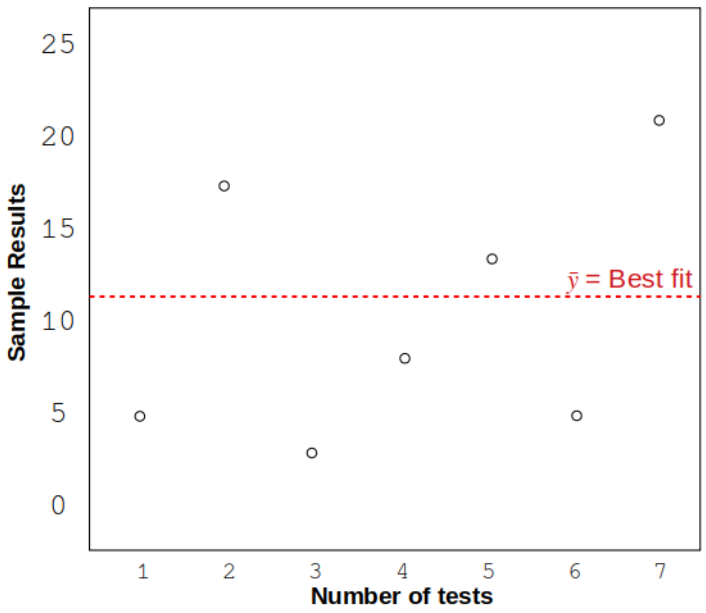
```
> head(rikz)
   Sample Richness Week angle1 angle2 exposure salinity temperature    NAP
3       3       13    1     65     96       10     29.4        17.5 -1.336
10     10       17    1    143     89        8     29.6        20.8 -1.334
2       2       10    1     62     96       10     29.4        17.5 -1.036
38     38        7    3     55     32       10     26.4        20.0 -1.005
11     11        6    2     41     42       11     27.9        15.8 -0.976
29     29        6    3     48     36       11     27.1        17.4 -0.893
   penetrability grainsize humus chalk sorting1 Beach
3          237.1     194.5  0.10  3.45   59.220     1
10         257.9     197.0  0.00  2.60   59.575     2
```

```
2          226.9     200.0  0.30  2.50   59.000     1
38         382.8     355.0  0.00 16.70  133.085     8
11         268.4     330.5  0.05  3.40  101.330     3
29         179.3     336.0  0.05 15.50  151.665     6


##### // Create 'rikz2' that only contains Beaches 1-4 from week 1 //

> names(rikz)
 [1] "Sample"        "Richness"      "Week"          "angle1"
 [5] "angle2"        "exposure"      "salinity"      "temperature"
 [9] "NAP"           "penetrability" "grainsize"     "humus"
[13] "chalk"         "sorting1"      "Beach"

> rikz2 = rikz[rikz$Week == 1,][rikz[rikz$Week == 1,]$Beach > 0
            & rikz[rikz$Week == 1,]$Beach < 5,]

> head(rikz2)
   Sample Richness Week angle1 angle2 exposure salinity temperature    NAP
3       3       13    1     65     96       10     29.4        17.5 -1.336
10     10       17    1    143     89        8     29.6        20.8 -1.334
2       2       10    1     62     96       10     29.4        17.5 -1.036
5       5       10    1     23     96       10     29.4        17.5 -0.684
1       1       11    1     32     96       10     29.4        17.5  0.045
9       9       19    1     26     89        8     29.6        20.8  0.061
   penetrability grainsize humus chalk sorting1 Beach
3          237.1     194.5  0.10  3.45   59.220     1
10         257.9     197.0  0.00  2.60   59.575     2
2          226.9     200.0  0.30  2.50   59.000     1
5          251.9     202.0  0.05  2.45   57.760     1
1          253.9     222.5  0.05  2.05   69.830     1
9          248.9     205.5  0.15  2.30   58.810     2


##### // The mean chalk value for Beach 5 //

> rikz[rikz$Beach == 5,][rikz[rikz$Beach == 5,]$chalk,]
     Sample Richness Week angle1 angle2 exposure salinity temperature    NAP
22       22       22    4     22     21       10     29.9        19.8 -0.503
25       25        6    4     18     21       10     29.9        19.8  0.054
25.1     25        6    4     18     21       10     29.9        19.8  0.054
     penetrability grainsize humus chalk sorting1 Beach
22           256.1     265.0   0.0   1.6   89.035     5
25           231.1     254.5   0.1   2.1   86.170     5
25.1         231.1     254.5   0.1   2.1   86.170     5

##### // The mean grain size for Beach 8 //

> mean(rikz[rikz$Beach == 8,][,'grainsize'])
[1] 327.9


##### // Save the new data.frame 'rikz_2.csv' //

> write.csv(rikz, file = "rikz_2.csv")
```

11. Linear Models, predictions and probability distributions

11.1 Some terms

- **Mean** - is a set of values is the quantity commonly called the mean or the average. To get the mean sum the number of of values and divide the result by the number of values.



# of tests	Sample Results
1	5
2	17
3	11
4	8
5	14
6	5
7	21

Mean = $\bar{y}$ = 11.57

$$\bar{y} = \frac{\sum x}{n}$$

Illustration 5: The mean

lm(y ~ 1) will return the mean value.

- **Regression** - A method for fitting a curve or a straight line through a set of points using some goodness-of-fit criterion. The most common type of regression is linear regression.
- **Linear Regression** - A regression that is linear in the unknown parameters used in the fit. It is defined by the ***intercept point (α)*** of the best fit line where the x axis is equal to zero plus the ***slope (β)*** of the best fit line times the value of x. The ***intercept point (α)*** is the mean.

- ***y = α +β * x***

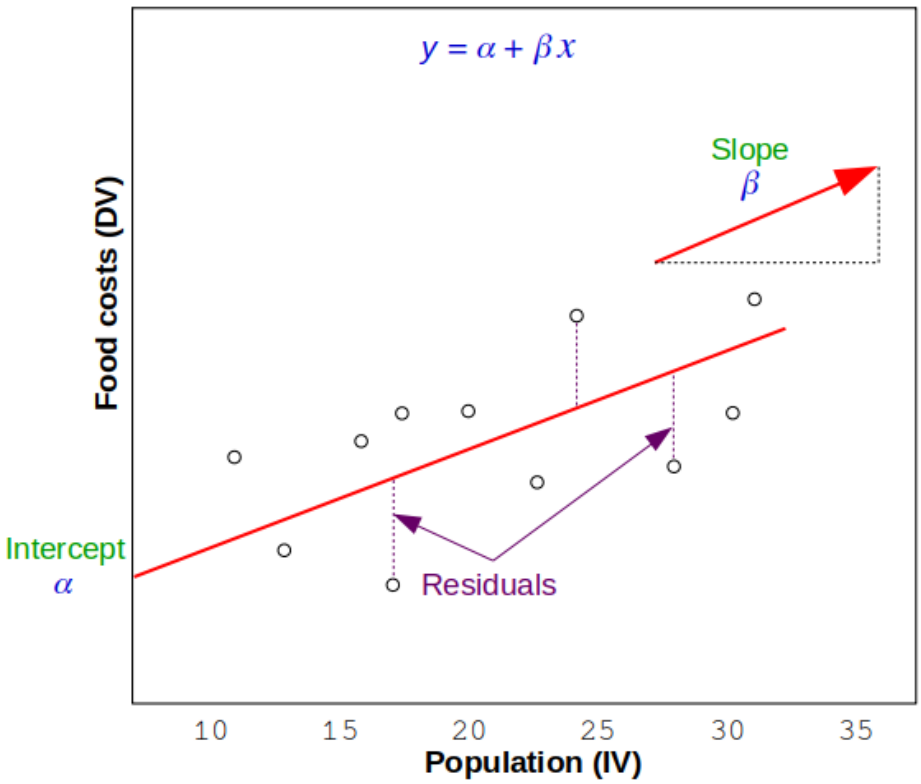


Illustration 6: Linear regression model

- **Multiple Regression** - A regression giving conditional expectation values of a given variable in terms of two or more other variables.
- **Null Hypothesis (H_0)** - A H_0 is a statistical hypothesis that is tested for possible rejection under the assumption that it is true (usually that observations are the result of chance). It means that there is no expectation of change.
- **Alternative Hypothesis (H_1)** - is the hypothesis used in hypothesis testing that is contrary to the H_0. It is usually taken to be that the observations are the result of a real effect (with some amount of chance variation superposed). In the p-test values less than 0.05 indicate a significance and indicate contrary to the H_0.
- **Estimate** - Mean expectation of the mean, the Null Hypothesis (H_0).
- **Standard Error** - This gives one standard deviation (**δ**) of the certainty of the accuracy of the Estimate (mean). 2δ gives the certainty of accuracy for 95% of values. mean = 6.7195 and 2δ = 0.55, therefore there is a 95% chance of values giving a mean in the range of 6.17 - 7.27.
- **t-value** - The coefficient divided by its standard error. The standard error is an estimate of the standard deviation of the coefficient, the amount it varies across cases. It can be thought of as a measure of the precision with which the regression coefficient is measured. This value says how many standard deviations from zero. For example:

  the *Estimate / Std. Error = t-value* ==> **6.7195 / 0.2726 = 24.64967**

- **p-value** - Shows the expectation that the *Estimate* is accurate for values. A small p-value (typically ≤ 0.05) indicates strong evidence against the H_0, so you reject the H_0. A large p-value (> 0.05) indicates weak evidence against the H_0, so you shouldn't reject H_0. The p-value of 0.05 means that there is less than a 5% chance of the results arising due to random chance.
- **t-test** - Refers to any statistical hypothesis test in which the test statistic follows a t-distribution under the H_0. A t-test is most commonly applied when the test statistic would follow a normal distribution if the value of a scaling term in the test statistic were known. Here is an example with the built-in *mtcars* dataset. It shows a t-test to compare vehicles in rows 1 to 10 with vehicles in rows 11 to 20 for column 1 which is miles per gallon (mpg). The H_0 is that there there is little to no statistical difference between the groups in terms of *mpg*. As the p-value from the test is 0.8744, it concurs with the H_0 and sets the expectation that there is indeed little statistical expectation of a significant difference.

```
> data(mtcars)

> head(mtcars[1])
                   mpg
Mazda RX4         21.0
Mazda RX4 Wag     21.0
Datsun 710        22.8
Hornet 4 Drive    21.4
Hornet Sportabout 18.7
Valiant           18.1

> a = t.test(mtcars[1:10,1],mtcars[11:20,1])

> print (a)

      Welch Two Sample t-test

data:  mtcars[1:10, 1] and mtcars[11:20, 1]
t = 0.16185, df = 10.892, p-value = 0.8744
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -6.055477  7.015477
sample estimates:
mean of x mean of y
    20.37     19.89

> str(a)
List of 9
 $ statistic  : Named num 0.162
  ..- attr(*, "names")= chr "t"
 $ parameter  : Named num 10.9
  ..- attr(*, "names")= chr "df"
 $ p.value    : num 0.874
 $ conf.int   : atomic [1:2] -6.06 7.02
  ..- attr(*, "conf.level")= num 0.95
 $ estimate   : Named num [1:2] 20.4 19.9
  ..- attr(*, "names")= chr [1:2] "mean of x" "mean of y"
 $ null.value : Named num 0
  ..- attr(*, "names")= chr "difference in means"
 $ alternative: chr "two.sided"
 $ method     : chr "Welch Two Sample t-test"
 $ data.name  : chr "mtcars[1:10, 1] and mtcars[11:20, 1]"
 - attr(*, "class")= chr "htest"
```

```
> unclass(a)
 $statistic
         t
 0.1618478

 $parameter
       df
 10.89198

 $p.value
 [1] 0.8743889

 $conf.int
 [1] -6.055477  7.015477
 attr(,"conf.level")
 [1] 0.95

 $estimate
 mean of x mean of y
     20.37     19.89

 $null.value
 difference in means
                   0

 $alternative
 [1] "two.sided"

 $method
 [1] "Welch Two Sample t-test"

 $data.name
 [1] "mtcars[1:10, 1] and mtcars[11:20, 1]"

 # Extract the mean of y
 > a$estimate[2]
 mean of y
     19.89

 # Extract the 'p' value

 > a$p.value
 [1] 0.8743889
```

- Dependent Variable (DV) - The variable being tested and measured in an experiment.
- Independent Variable (IV) - The variable that is changed or controlled in a scientific experiment to test the effects on the DV.
- Analysis of Variance (ANOVA) - is a collection of statistical models and their associated estimation procedures used to analyse the differences among group means in a sample.

$$y = \alpha + \beta x_1 + \beta x_2$$

- Multivarite ANOVA (MANOVA) - is a procedure for comparing multivariate sample means. As a multivariate procedure, it is used when there are two or more dependent variables, and is typically followed by significance tests involving individual dependent variables separately.

Analysis of covariance (ANCOVA) - is a general linear model which blends ANOVA and regression. ANCOVA evaluates whether the means of a DV are equal across levels of a categorical IV.

- Generalised Linear Model (GLM) is a flexible generalisation of ordinary linear regression that allows for response variables that have error distribution models other than a normal distribution.
- Generalised Linear Mixed Model (GLMM) - is an extension to GLM in which the linear predictor contains random effects in addition to the usual fixed effects. They also inherit from GLMs the idea of extending linear mixed models to non-normal data.

11.2 Demonstration

Working with the *owl_data.csv* file once more to demonstrate linear models, predictions and probability distributions.

```
> setwd('~/datasets/owl_data_2')

> owl = read.csv('owl_data.csv')

##### // Inspect the data //

> str(owl)
'data.frame': 599 obs. of  5 variables:
 $ nest   : Factor w/ 27 levels "AutavauxTV","Bochet",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ sex    : Factor w/ 2 levels "Female","Male": 2 2 2 2 2 2 1 2 1 ...
 $ food   : num  22.2 22.4 22.5 22.6 22.6 ...
 $ begging: int  4 0 2 2 2 2 18 4 18 0 ...
 $ brood  : int  5 5 5 5 5 5 5 5 5 5 ...

> summary(owl)
          nest         sex           food          begging          brood
 Oleyes     : 52   Female:245   Min.   :21.71   Min.   : 0.00   Min.   :1.000
 Moutet     : 41   Male  :354   1st Qu.:23.11   1st Qu.: 0.00   1st Qu.:4.000
 Etrabloz   : 34                Median :24.38   Median : 5.00   Median :4.000
 Yvonnand   : 34                Mean   :24.76   Mean   : 6.72   Mean   :4.392
 Champmartin: 30                3rd Qu.:26.25   3rd Qu.:11.00   3rd Qu.:5.000
 Lucens     : 29                Max.   :29.25   Max.   :32.00   Max.   :7.000
 (Other)    :379

> head(owl)
        nest  sex  food begging brood
1 AutavauxTV Male 22.25       4     5
2 AutavauxTV Male 22.38       0     5
3 AutavauxTV Male 22.53       2     5
4 AutavauxTV Male 22.56       2     5
5 AutavauxTV Male 22.61       2     5
6 AutavauxTV Male 22.65       2     5
```

The purpose of this is to explain the *begging rate* of the chicks in the nest.

Response variable = y = begging

lm(): is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance.

11.3 The mean()

Question 1: What is the begging rate of the sampled population?

```
mean(owl$begging)
[1] 6.719533

> lm(begging ~ 1, data = owl)

Call:
lm(formula = begging ~ 1, data = owl)

Coefficients:
(Intercept)
       6.72
```

lm(y ~ 1) returns the mean value (**α**), the intercept where ***βx = 1***.

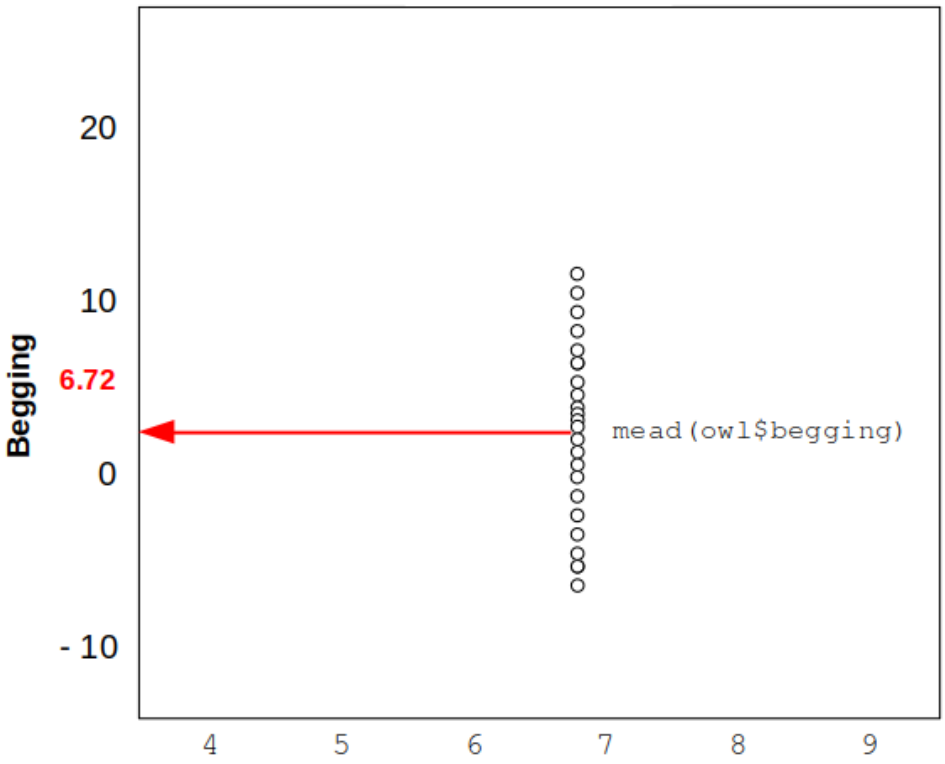


Illustration 7: Mean begging

Now the statistics are given in the *summary()* of the data. *Estimate*, *Standard error*, *t-value* and *p-value*.

```
> mod = lm(begging ~ 1, data = owl)
```

```
> summary(mod)

Call:
lm(formula = begging ~ 1, data = owl)

Residuals:
   Min     1Q Median     3Q    Max
 -6.72  -6.72  -1.72   4.28  25.28

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   6.7195     0.2726   24.65   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.671 on 598 degrees of freedom
```

11.4 The t-test

Question 2: What is the begging rate of males versus females?

In this case *R* coded *females = 0* and *males = 1*. So therefore females indicate the intercept (* 0) and males the difference from the intercept. This is indicated by *sexMale* in the second line of the output, males have been labelled internally as = 1, therefore *sexFemale* must be = 0. So the intercept of females 6.0122 plus the difference 1.1968 gives the intercept value for males = 7.209.

In *R* the linear modelling is carried out using the *lm()* function. The DV is placed to the left side of the tilde and the IV or IVs are placed to the right. As such the DV are the *y* axis and the IVs are on the *x* axis of associated graphs.

lm(DV ~ IV(s), data=<dataset>)

```
> lm(begging ~ sex, data = owl)

Call:
lm(formula = begging ~ sex, data = owl)

Coefficients:
(Intercept)      sexMale
      6.012        1.197

> mod2 = lm(begging ~ sex, data = owl)
```

```
> summary(mod2)

Call:
lm(formula = begging ~ sex, data = owl)

Residuals:
   Min     1Q Median     3Q    Max
-7.209 -6.012 -1.209  4.791 24.791

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   6.0122     0.4249  14.151   <2e-16 ***
sexMale       1.1968     0.5527   2.165   0.0307 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.65 on 597 degrees of freedom
Multiple R-squared:  0.007793,        Adjusted R-squared:  0.006131
F-statistic: 4.689 on 1 and 597 DF,  p-value: 0.03075
```

lm(y ~ x1 + x2 + ...)

$$y = \alpha + \beta x_1 + \beta x_2 + \beta x_n$$

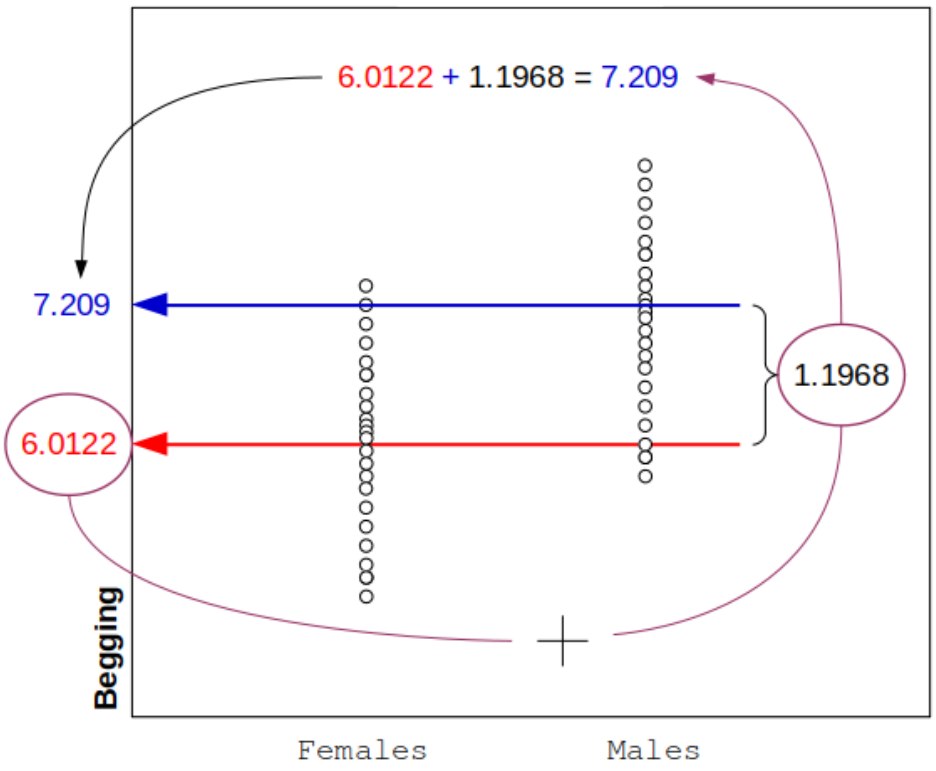


Illustration 8: Mean begging rate based on sex

11.5 ANOVA, the Analysis of Variance

Comparing the means of greater than two groups.

Question 3: How does the begging rate change as nest size increases?

Important: First change *owl$brood* to a factor.

```
> str(owl$brood)
 int [1:599] 5 5 5 5 5 5 5 5 5 5 ...

> owl$brood = as.factor(owl$brood)

> str(owl$brood)
 Factor w/ 7 levels "1","2","3","4",..: 5 5 5 5 5 5 5 5 5 5 ...
```

This is an ANOVA. the *lm(begging ~ brood, data = owl)* doesn't give much information but the *summary(lm(begging ~ brood, data = owl))* gives more useful information with *brood1* indicating the intercept and all other broods are indicated as the difference from *brood1*.

Can't be certain that the birds didn't beg at a negative rate. Obviously this isn't possible.

t-value and p-values demonstrate the certainty of difference between each brood and *brood1*. However there is a high level of uncertainty of H_0.

For example *brood7* p-value is less than 0.05 and therefore indicates that it doesn't have the same begging rate as *brood1*, i.e. reject the H_0.

This test is only comparing each brood relative to *brood1*. To review this further post-hoc tests which will carry out statistics between each brood group.

```
> lm(begging ~ brood, data = owl)

Call:
lm(formula = begging ~ brood, data = owl)

Coefficients:
(Intercept)       brood2       brood3       brood4       brood5       brood6
     4.0000      -0.5263       0.4237       3.0905       2.6404       4.5385
     brood7
     7.5000

> mod3 = lm(begging ~ brood, data = owl)
```

```
> summary(mod3)

Call:
lm(formula = begging ~ brood, data = owl)

Residuals:
    Min      1Q  Median      3Q     Max
-11.500  -5.090  -1.500   4.743  25.360

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   4.0000     3.2477   1.232   0.2186
brood2       -0.5263     3.4144  -0.154   0.8775
brood3        0.4237     3.3560   0.126   0.8996
brood4        3.0905     3.2785   0.943   0.3462
brood5        2.6404     3.2761   0.806   0.4206
brood6        4.5385     3.4886   1.301   0.1938
brood7        7.5000     3.4334   2.184   0.0293 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.495 on 592 degrees of freedom
Multiple R-squared:  0.06141, Adjusted R-squared:  0.0519
F-statistic: 6.456 on 6 and 592 DF,  p-value: 1.317e-06
```

lm(y ~ x1 + x2 + ...)

$$y = \alpha + \beta x_1 + \beta x_2 + \beta x_n$$

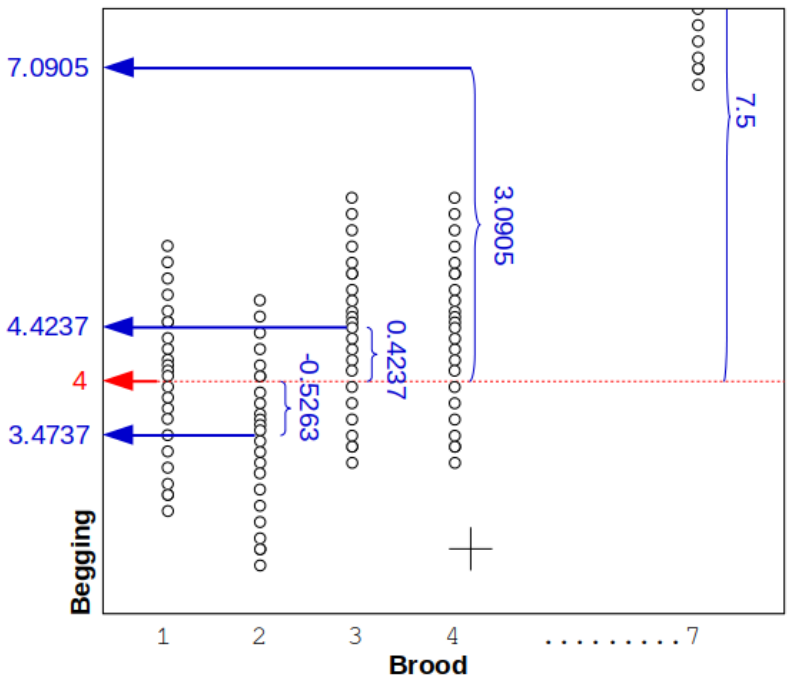


Illustration 9: ANOVA

anova() - Compute analysis of variance (or deviance) tables for one or more fitted model objects.

aov() - Fit an analysis of variance model by a call to *lm* for each stratum.

tukey() - Create a set of confidence intervals on the differences between the means of the levels of a factor with the specified family-wise probability of coverage. The intervals are based on the Studentised range statistic, *Tukey Honest Significant Difference* method.

```
> post = aov(mod3)

> TukeyHSD(post)
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = mod3)

$brood
          diff         lwr       upr     p adj
2-1 -0.5263158 -10.6277301  9.575099 0.9999989
3-1  0.4237288  -9.5049990 10.352457 0.9999997
4-1  3.0904762  -6.6089619 12.789914 0.9654242
5-1  2.6403509  -7.0519281 12.332630 0.9843572
6-1  4.5384615  -5.7825744 14.859498 0.8514925
7-1  7.5000000  -2.6578473 17.657847 0.3055212
3-2  0.9500446  -3.0470773  4.947167 0.9923990
4-2  3.6167920   0.2291015  7.004482 0.0276208
5-2  3.1666667  -0.2004714  6.533805 0.0809328
6-2  5.0647773   0.1738541  9.955701 0.0368230
7-2  8.0263158   3.4898846 12.562747 0.0000048
4-3  2.6667474  -0.1647738  5.498269 0.0800918
5-3  2.2166221  -0.5902774  5.023521 0.2284480
6-3  4.1147327  -0.4087831  8.638249 0.1022524
7-3  7.0762712   2.9386060 11.213936 0.0000116
5-4 -0.4501253  -2.2880991  1.387848 0.9910760
6-4  1.4479853  -2.5472191  5.443190 0.9360239
7-4  4.4095238   0.8570970  7.961951 0.0048667
6-5  1.8981107  -2.0796816  5.875903 0.7955024
7-5  4.8596491   1.3268162  8.392482 0.0010456
7-6  2.9615385  -2.0448994  7.967976 0.5824679
```

11.6 Regression

Compare the means of a continuous variable.

Question 4: How does begging rate change as food increases?

This demonstrates that the expectation line intercepts the *y* axis at a 26.9 begging rate. This may or may not have an actual biological meaning as the samples taken may not actually have gone down to zero food. The same can be said of the line crossing the *x* axis. However it is useful mathematically. The slope of negative (-0.81) means the expectation is that *begging* reduces by 0.81 for every 1 unit of *food* added. The very low *p-value* means that there is a very high confidence that more food decreases the begging rate.

```
> lm(begging ~ food, data = owl)

Call:
lm(formula = begging ~ food, data = owl)

Coefficients:
(Intercept)         food
    26.9738      -0.8181

> mod4 = lm(begging ~ food, data = owl)
```

```
> summary(mod4)

Call:
lm(formula = begging ~ food, data = owl)

Residuals:
   Min     1Q Median     3Q    Max
-9.155 -5.253 -1.298  4.454 25.544

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  26.9738     3.4431   7.834 2.17e-14 ***
food         -0.8181     0.1387  -5.900 6.09e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.49 on 597 degrees of freedom
Multiple R-squared:  0.0551,  Adjusted R-squared:  0.05351
F-statistic: 34.81 on 1 and 597 DF,  p-value: 6.092e-09
```

$$lm(y \sim x1 + x2 + ...)$$

$$y = \alpha + \beta x_1 + \beta x_2 + \beta x_n$$

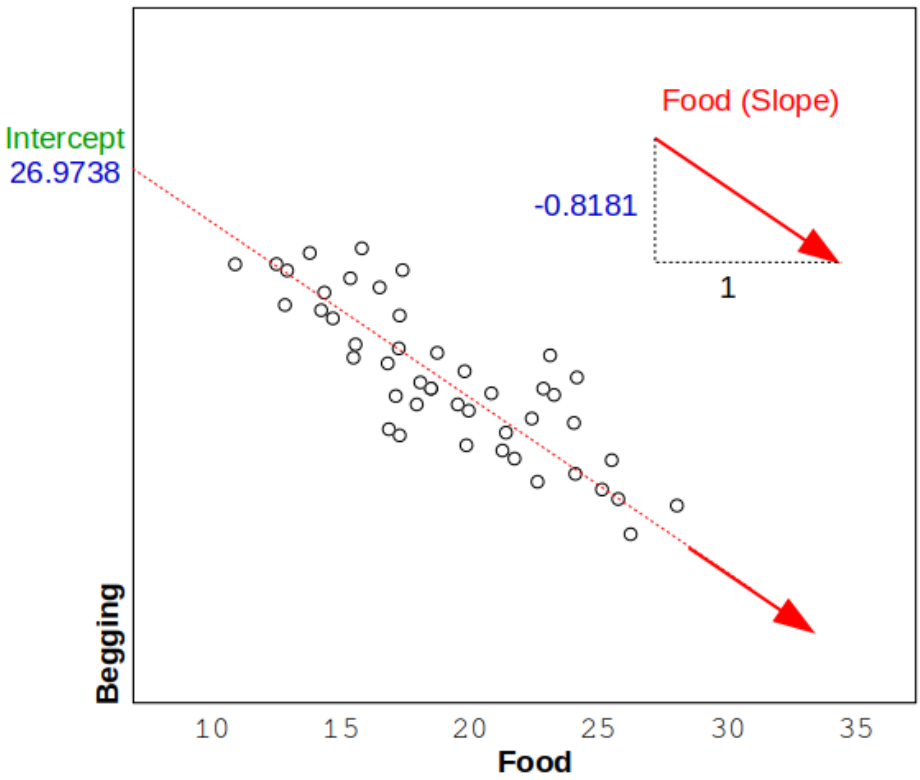


Illustration 10: Regression

11.7 Multiple regression

Compare the means of a continuous variable plus a group variable.

Question 5a: How does begging rate change as food and sex increase?

```
> lm(begging ~ food + sex, data = owl)

Call:
lm(formula = begging ~ food + sex, data = owl)

Coefficients:
(Intercept)         food      sexMale
    26.4475      -0.8276       1.2892

> mod5a = lm(begging ~ food + sex, data = owl)

> summary(mod5a)

Call:
lm(formula = begging ~ food + sex, data = owl)

Residuals:
   Min     1Q Median     3Q    Max
-9.711 -5.244 -1.597  4.700 25.020

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  26.4475     3.4365   7.696 5.85e-14 ***
food         -0.8276     0.1382  -5.990 3.63e-09 ***
sexMale       1.2892     0.5374   2.399   0.0168 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.464 on 596 degrees of freedom
Multiple R-squared:  0.06413, Adjusted R-squared:  0.06099
F-statistic: 20.42 on 2 and 596 DF,  p-value: 2.642e-09
```

$$\text{lm(y ~ x1 + x2 + ...)}$$

$$y = \alpha + \beta x_1 + \beta x_2 + \beta x_n$$

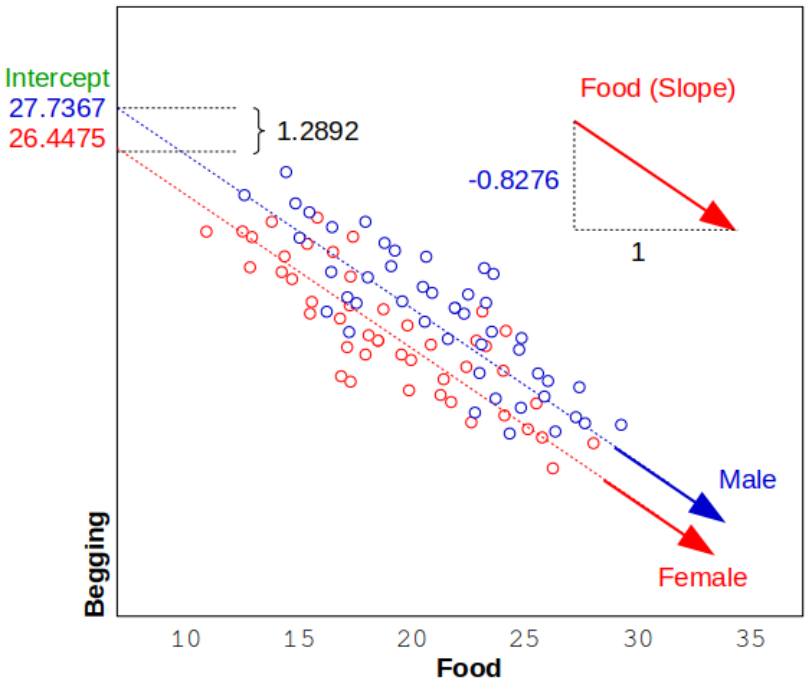


Illustration 11: Multiple Regression

Question 5b: How does begging rate change as food and brood increase?

The data is a multidimensional model with both food and brood. In this case brood is now continuous data showing brood size (as.numeric(owl$brood)).

```
> owl$brood = as.numeric(owl$brood)

> lm(begging ~ food + brood, data = owl)

Call:
lm(formula = begging ~ food + brood, data = owl)

Coefficients:
(Intercept)         food        brood
    21.0447      -0.7877       1.1786

> mod5b = lm(begging ~ food + brood, data = owl)

> summary(mod5b)

Call:
lm(formula = begging ~ food + brood, data = owl)

Residuals:
    Min      1Q  Median      3Q     Max
-10.981  -5.250  -1.300   4.514  24.818

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  21.0447     3.5561   5.918 5.50e-09 ***
food         -0.7877     0.1358  -5.799 1.08e-08 ***
brood         1.1786     0.2258   5.220 2.47e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.352 on 596 degrees of freedom
Multiple R-squared:  0.09641, Adjusted R-squared:  0.09338
F-statistic: 31.79 on 2 and 596 DF,  p-value: 7.583e-14
```

$$lm(y \sim x1 + x2 + ...)$$

$$y = \alpha + \beta x_1 + \beta x_2 + \beta x_n$$

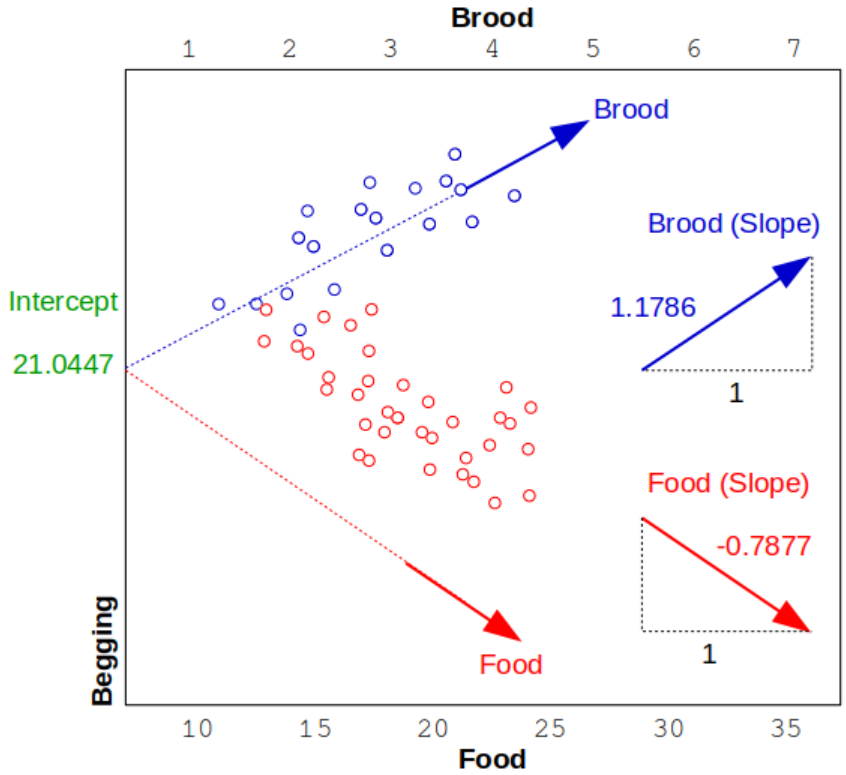


Illustration 12: Another multiple regression

11.8 ANCOVA, the Analysis of Covariance

Comparing the means of multiple variables and allow for interactions between them.

Question 5c: How does the begging rate change as food & sex increase, while allowing for the food effect of each sex to differ?

The interaction of food & sex: an ANCOVA. If the effect of both food and sex change relative to each other then the *interaction* of both must be accounted for. This allows the relationship between males and females to change. As the *p-value* is quite large there is no reason to reject the H_0. In other words there is little evidence that there is a divergence between the begging rate of males and females given greater levels of food.

```
> lm(begging ~ food + sex + food*sex, data = owl)

Call:
lm(formula = begging ~ food + sex + food * sex, data = owl)

Coefficients:
 (Intercept)          food       sexMale  food:sexMale
     23.6642       -0.7149        6.0699       -0.1933

> mod5c = lm(begging ~ food + sex + food*sex, data = owl)

> summart(mod5c)
Error in summart(mod5c) : could not find function "summart"
```

```
> summary(mod5c)

Call:
lm(formula = begging ~ food + sex + food * sex, data = owl)

Residuals:
   Min     1Q Median     3Q    Max
-9.955 -5.241 -1.461  4.515 25.042

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   23.6642     5.3035   4.462 9.71e-06 ***
food          -0.7149     0.2141  -3.338 0.000895 ***
sexMale        6.0699     6.9569   0.873 0.383286
food:sexMale  -0.1933     0.2804  -0.689 0.490936
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.467 on 595 degrees of freedom
Multiple R-squared:  0.06488, Adjusted R-squared:  0.06016
F-statistic: 13.76 on 3 and 595 DF,  p-value: 1.094e-08
```

lm(y ~ x1 + x2 + x1 * x2)

$$y = \alpha + \beta x_1 + \beta x_2 + \beta x_1 * \beta x_2$$

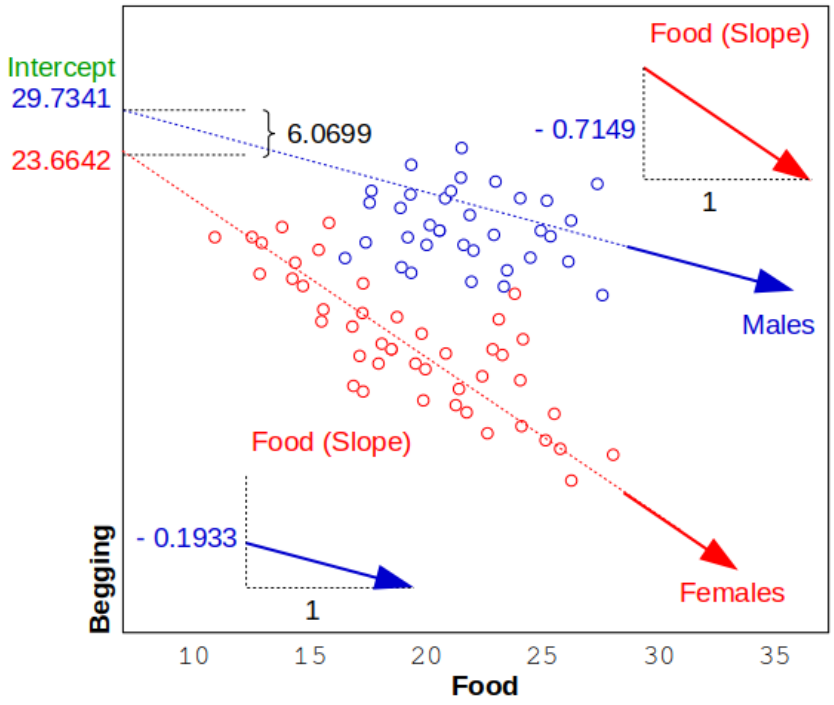


Illustration 13: ANCOVA

11.9 Linear model summary

Function	R function	Result
$y = \alpha + \beta x$	lm(begging ~ 1)	The mean of y
$y = \alpha + \beta x$	lm(begging ~ sex)	t-test
$y = \alpha + \beta x_1 + \beta x_2$	lm(begging ~ brood size)	ANOVA
$y = \alpha + \beta x$	lm(begging ~ food)	Simple regression
$y = \alpha + \beta x_1 + \beta x_2$	lm(begging ~ food + sex)	Multiple regression
$y = \alpha + \beta x_1 + \beta x_2$	lm(begging ~ food + brood)	Multiple regression
$y = \alpha + \beta x_1 + \beta x_2$	lm(begging ~ food + sex + food*sex)	ANCOVA

All linear models are about calculating the mean of a particular group or groups. The *statistics* simply tell you how confident you are that the means differ between groups.

11.10 Exercise: Linear models

1. Import the *RIKZ.csv* data into an object called rikz (inspect the data etc).
2. What is the average species richness across all samples?
3. How does species richness vary with sand grainsize?
    - HINT: simple regression.
4. How does species richness vary at the different beaches?
    - HINT: ANOVA.
5. How does species richness vary with sampling time (week 1&2 versus week 3&4)?
    - HINT: t-test (will need to create a 2-category variable from *week*).
6. How does species richness vary with NAP and beach angle; is there any evidence of an interaction effect between them?
    - HINT: multiple regression.

Question 1: Import the 'RIKZ.csv' data into an object called rikz.

```
> setwd('~/datasets/RIKZ_2')

> rikz = read.csv('RIKZ.csv')
```

Explore the data frame.

```
> summary(rikz)
     Sample      Richness           Week           angle1           angle2
 Min.   : 1   Min.   : 0.000   Min.   :1.000   Min.   :  6.00   Min.   :21.00
 1st Qu.:12   1st Qu.: 3.000   1st Qu.:2.000   1st Qu.: 22.00   1st Qu.:32.00
 Median :23   Median : 4.000   Median :2.000   Median : 32.00   Median :42.00
 Mean   :23   Mean   : 5.689   Mean   :2.333   Mean   : 50.31   Mean   :57.78
 3rd Qu.:34   3rd Qu.: 8.000   3rd Qu.:3.000   3rd Qu.: 55.00   3rd Qu.:89.00
 Max.   :45   Max.   :22.000   Max.   :4.000   Max.   :312.00   Max.   :96.00
    exposure        salinity     temperature         NAP
 Min.   : 8.00   Min.   :26.4   Min.   :15.80   Min.   :-1.3360
 1st Qu.:10.00   1st Qu.:27.1   1st Qu.:17.50   1st Qu.:-0.3750
 Median :10.00   Median :27.9   Median :18.77   Median : 0.1670
 Mean   :10.22   Mean   :28.1   Mean   :18.77   Mean   : 0.3477
 3rd Qu.:11.00   3rd Qu.:29.4   3rd Qu.:20.00   3rd Qu.: 1.1170
 Max.   :11.00   Max.   :29.9   Max.   :20.80   Max.   : 2.2550
 penetrability     grainsize         humus             chalk
 Min.   :151.8   Min.   :186.0   Min.   :0.00000   Min.   : 0.850
 1st Qu.:237.1   1st Qu.:222.5   1st Qu.:0.00000   1st Qu.: 2.200
 Median :256.1   Median :266.0   Median :0.05000   Median : 4.750
 Mean   :289.4   Mean   :272.5   Mean   :0.05028   Mean   : 7.961
 3rd Qu.:272.9   3rd Qu.:316.5   3rd Qu.:0.10000   3rd Qu.: 9.150
 Max.   :624.0   Max.   :405.5   Max.   :0.30000   Max.   :45.750
    sorting1          Beach
 Min.   : 53.08   Min.   :1
 1st Qu.: 72.75   1st Qu.:3
 Median : 89.03   Median :5
 Mean   : 97.82   Mean   :5
 3rd Qu.:115.44   3rd Qu.:7
 Max.   :248.60   Max.   :9

> str(rikz)
'data.frame': 45 obs. of  15 variables:
 $ Sample       : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Richness     : int  11 10 13 11 10 8 9 8 19 17 ...
 $ Week         : int  1 1 1 1 1 1 1 1 1 1 ...
 $ angle1       : int  32 62 65 55 23 129 126 52 26 143 ...
 $ angle2       : int  96 96 96 96 96 89 89 89 89 89 ...
 $ exposure     : int  10 10 10 10 10 8 8 8 8 8 ...
 $ salinity     : num  29.4 29.4 29.4 29.4 29.4 29.6 29.6 29.6 29.6 29.6 ...
 $ temperature  : num  17.5 17.5 17.5 17.5 17.5 20.8 20.8 20.8 20.8 20.8 ...
 $ NAP          : num  0.045 -1.036 -1.336 0.616 -0.684 ...
 $ penetrability: num  254 227 237 249 252 ...
 $ grainsize    : num  222 200 194 221 202 ...
 $ humus        : num  0.05 0.3 0.1 0.15 0.05 0.1 0.1 0.1 0.15 0 ...
 $ chalk        : num  2.05 2.5 3.45 1.6 2.45 2.5 1.85 1.7 2.3 2.6 ...
 $ sorting1     : num  69.8 59 59.2 67.8 57.8 ...
 $ Beach        : int  1 1 1 1 1 2 2 2 2 2 ...

> names(rikz)
 [1] "Sample"        "Richness"      "Week"          "angle1"
 [5] "angle2"        "exposure"      "salinity"      "temperature"
 [9] "NAP"           "penetrability" "grainsize"     "humus"
[13] "chalk"         "sorting1"      "Beach"
```

```
> head(rikz)
  Sample Richness Week angle1 angle2 exposure salinity temperature    NAP
1      1       11    1     32     96       10     29.4        17.5  0.045
2      2       10    1     62     96       10     29.4        17.5 -1.036
3      3       13    1     65     96       10     29.4        17.5 -1.336
4      4       11    1     55     96       10     29.4        17.5  0.616
5      5       10    1     23     96       10     29.4        17.5 -0.684
6      6        8    1    129     89        8     29.6        20.8  1.190
  penetrability grainsize humus chalk sorting1 Beach
1         253.9     222.5  0.05  2.05   69.830     1
2         226.9     200.0  0.30  2.50   59.000     1
3         237.1     194.5  0.10  3.45   59.220     1
4         248.6     221.0  0.15  1.60   67.750     1
5         251.9     202.0  0.05  2.45   57.760     1
6         250.1     192.5  0.10  2.50   53.075     2
```

Question 2: What is the average species richness across all samples?

```
Mean   : 5.689   # Extracted from the summary(rikz)

# Alternatively extract from the intercept (α) where y = 0

> lm (Richness ~ 1, rikz)

Call:
lm(formula = Richness ~ 1, data = rikz)

Coefficients:
(Intercept)
      5.689
```

Question 3: How does species richness vary with sand grainsize?

```
> rikz 1 = lm(Richness ~ grainsize, data=rikz)

> summary(rikz1)

Call:
lm(formula = Richness ~ grainsize, data = rikz)

Residuals:
    Min      1Q  Median      3Q     Max
-8.4833 -3.6733 -0.2693  2.1872 16.0701

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 14.49529    3.40326   4.259 0.000109 ***
grainsize   -0.03232    0.01222  -2.644 0.011386 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.694 on 43 degrees of freedom
Multiple R-squared:  0.1399,  Adjusted R-squared:  0.1198
F-statistic: 6.991 on 1 and 43 DF,  p-value: 0.01139
```

Answer, Very little as the slope of the line is almost zero with a p-value < 0.05 means that the model demonstrates a high probability of H_0.

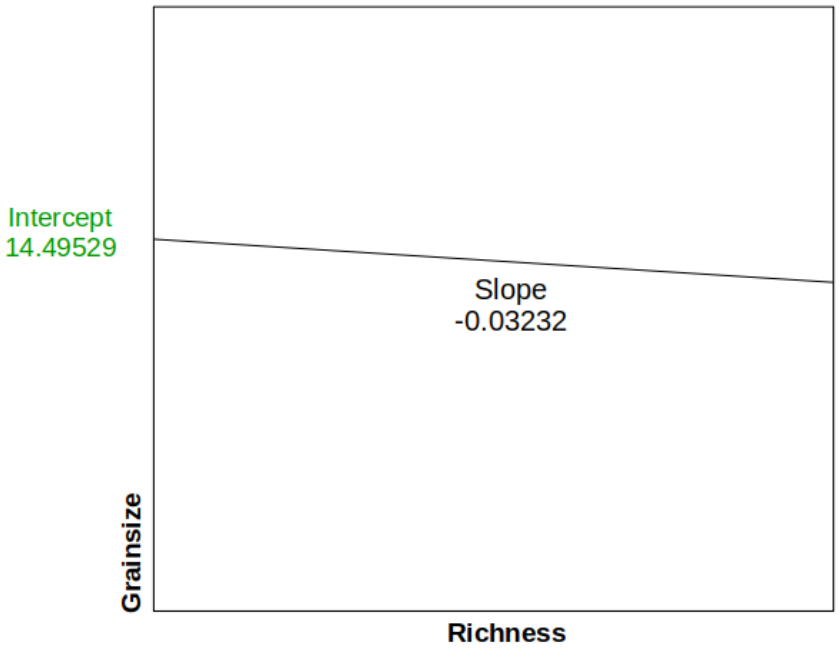


Illustration 14: Richness/grainsize

Question 4: How does species richness vary at the different beaches?

```
> rikz$Beach = as.factor(rikz$Beach)

> rikz2 = lm(Richness ~ Beach, data=rikz)

> summary(rikz2)

Call:
lm(formula = Richness ~ Beach, data = rikz)

Residuals:
   Min     1Q Median     3Q    Max
  -7.4   -1.4   -0.2    1.0   14.6

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   11.000      1.761   6.245 3.27e-07 ***
Beach2         1.200      2.491   0.482  0.63289
Beach3        -7.600      2.491  -3.051  0.00426 **
Beach4        -8.600      2.491  -3.453  0.00144 **
Beach5        -3.600      2.491  -1.445  0.15703
Beach6        -7.000      2.491  -2.810  0.00796 **
Beach7        -8.800      2.491  -3.533  0.00115 **
Beach8        -7.000      2.491  -2.810  0.00796 **
Beach9        -6.400      2.491  -2.569  0.01448 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.938 on 36 degrees of freedom
Multiple R-squared:  0.4931,  Adjusted R-squared:  0.3805
F-statistic: 4.378 on 8 and 36 DF,  p-value: 0.0009179
```

As the p-value is less than 0.05 there is a high level of confidence that at least one of the pairs of beaches vary from each other in species richness. Carrying out fit an analysis of variance model test (*aov()*) test to investigate further.

```
> rikz3 = aov(rikz2)

> summary (rikz3)
            Df Sum Sq Mean Sq F value   Pr(>F)
Beach        8  543.2   67.91   4.378 0.000918 ***
Residuals   36  558.4   15.51
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The resulting p-value of 0.000918 says that with a high level of probability that there is a variance between beaches. Compute the *Tukey Honest Significant Differences* between the beaches.

```
> rikz4 = TukeyHSD(rikz3)

> print(rikz4)
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = rikz2)

$Beach
              diff       lwr         upr     p adj
2-1  1.200000e+00  -7.01263  9.41262978 0.9999026
3-1 -7.600000e+00 -15.81263  0.61262978 0.0882779
4-1 -8.600000e+00 -16.81263 -0.38737022 0.0342201
5-1 -3.600000e+00 -11.81263  4.61262978 0.8723088
6-1 -7.000000e+00 -15.21263  1.21262978 0.1475693
7-1 -8.800000e+00 -17.01263 -0.58737022 0.0279853
8-1 -7.000000e+00 -15.21263  1.21262978 0.1475693
9-1 -6.400000e+00 -14.61263  1.81262978 0.2347476
3-2 -8.800000e+00 -17.01263 -0.58737022 0.0279853
4-2 -9.800000e+00 -18.01263 -1.58737022 0.0097691
5-2 -4.800000e+00 -13.01263  3.41262978 0.6005725
6-2 -8.200000e+00 -16.41263  0.01262978 0.0506097
7-2 -1.000000e+01 -18.21263 -1.78737022 0.0078509
8-2 -8.200000e+00 -16.41263  0.01262978 0.0506097
9-2 -7.600000e+00 -15.81263  0.61262978 0.0882779
4-3 -1.000000e+00  -9.21263  7.21262978 0.9999757
5-3  4.000000e+00  -4.21263 12.21262978 0.7953899
6-3  6.000000e-01  -7.61263  8.81262978 0.9999995
7-3 -1.200000e+00  -9.41263  7.01262978 0.9999026
8-3  6.000000e-01  -7.61263  8.81262978 0.9999995
9-3  1.200000e+00  -7.01263  9.41262978 0.9999026
5-4  5.000000e+00  -3.21263 13.21262978 0.5484761
6-4  1.600000e+00  -6.61263  9.81262978 0.9991816
7-4 -2.000000e-01  -8.41263  8.01262978 1.0000000
8-4  1.600000e+00  -6.61263  9.81262978 0.9991816
9-4  2.200000e+00  -6.01263 10.41262978 0.9925552
6-5 -3.400000e+00 -11.61263  4.81262978 0.9034383
7-5 -5.200000e+00 -13.41263  3.01262978 0.4968860
8-5 -3.400000e+00 -11.61263  4.81262978 0.9034383
9-5 -2.800000e+00 -11.01263  5.41262978 0.9664876
7-6 -1.800000e+00 -10.01263  6.41262978 0.9981055
8-6  8.881784e-16  -8.21263  8.21262978 1.0000000
9-6  6.000000e-01  -7.61263  8.81262978 0.9999995
8-7  1.800000e+00  -6.41263 10.01262978 0.9981055
9-7  2.400000e+00  -5.81263 10.61262978 0.9869248
9-8  6.000000e-01  -7.61263  8.81262978 0.9999995
```

Looking over the p-values there are a number of beach pairs that how a value lower than 0.05, in other words there is a difference in species diversity for these beach pairs. Convert the list to a dataframe and pull out that data.

```
> df.rikz4 = data.frame(unclass(rikz4))

> names(df.rikz4)
[1] "Beach.diff"  "Beach.lwr"   "Beach.upr"   "Beach.p.adj"

> df.rikz4[df.rikz4$Beach.p.adj < 0.05,]
    Beach.diff Beach.lwr  Beach.upr Beach.p.adj
4-1       -8.6 -16.81263 -0.3873702 0.034220065
7-1       -8.8 -17.01263 -0.5873702 0.027985274
3-2       -8.8 -17.01263 -0.5873702 0.027985274
4-2       -9.8 -18.01263 -1.5873702 0.009769101
7-2      -10.0 -18.21263 -1.7873702 0.007850934
```

There is a significant difference in species diversity between beaches:

- beach 4 and beach 1
- beach 7 and beach 1
- beach 3 and beach 2
- beach 4 and beach 2
- beach 7 and beach 2

Question 5: How does species richness vary with sampling time?

Week 1&2 versus week 3&4 t-test (need to create a 2-category variable from 'week'').

```
> rikz5 = rikz[rikz$Week < 5,]

> rikz5$fortnight = ifelse(rikz5$Week < 3, 1, 2)

> rikz5$fortnight = as.factor(rikz5$fortnight)

> rikz6 = lm(Richness ~ fortnight, data=rikz5)

> summary(rikz6)

Call:
lm(formula = Richness ~ fortnight, data = rikz5)

Residuals:
   Min     1Q Median     3Q    Max
 -5.24  -3.24  -1.00   2.00  17.00

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    6.240      1.004   6.212 1.79e-07 ***
fortnight2    -1.240      1.507  -0.823    0.415
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.022 on 43 degrees of freedom
Multiple R-squared:  0.01551, Adjusted R-squared:  -0.007387
F-statistic: 0.6774 on 1 and 43 DF,  p-value: 0.415
```

The p-value of $1.79e^{-07}$ being the intercept does not mean anything significant. The fact that 0.415 is greater that 0.05 indicates that there is little likelihood that there is any difference between species richness *fortnight1* and *fortnight2*.

Question 6: How does species richness vary with NAP and beach angle?

```
> rikz7 = lm(Richness ~ NAP + angle1, data=rikz)

> summary(rikz7)

Call:
lm(formula = Richness ~ NAP + angle1, data = rikz)

Residuals:
    Min      1Q  Median      3Q     Max
-4.9890 -3.0331 -0.7893  1.5244 14.0487

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  6.376753   0.903875   7.055 1.21e-08 ***
NAP         -2.862905   0.636315  -4.499 5.31e-05 ***
angle1       0.006113   0.012145   0.503    0.617
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.197 on 42 degrees of freedom
Multiple R-squared:  0.3286,  Adjusted R-squared:  0.2966
F-statistic: 10.28 on 2 and 42 DF,  p-value: 0.0002327


> rikz8 = lm(Richness ~ NAP + angle2, data=rikz)

> summary(rikz8)

Call:
lm(formula = Richness ~ NAP + angle2, data = rikz)

Residuals:
    Min      1Q  Median      3Q     Max
-5.4611 -2.4319 -0.8159  1.4524 15.7456

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.88471    1.34484   2.889  0.00609 **
NAP         -2.72332    0.60297  -4.517 5.03e-05 ***
angle2       0.04761    0.02024   2.353  0.02339 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.956 on 42 degrees of freedom
Multiple R-squared:  0.4032,  Adjusted R-squared:  0.3748
F-statistic: 14.19 on 2 and 42 DF,  p-value: 1.961e-05
```

The p-value for *angle1* is not significant however the *p-value* of 0.02339 for *angle2* demonstrates significance in the difference in species richness.

Question 7: Is there any evidence of an interaction effect between them?

```
> rikz9 = lm(Richness ~ NAP + angle1 + NAP * angle1, data=rikz)

> summary(rikz9)

Call:
lm(formula = Richness ~ NAP + angle1 + NAP * angle1, data = rikz)

Residuals:
    Min      1Q  Median      3Q     Max
-5.3839 -2.7098 -0.8666  1.6629 14.6538

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.68941    1.13138   5.029 1.02e-05 ***
NAP         -2.13058    0.96476  -2.208   0.0329 *
angle1       0.01947    0.01795   1.084   0.2846
NAP:angle1  -0.01418    0.01404  -1.010   0.3186
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.196 on 41 degrees of freedom
Multiple R-squared:  0.3449,  Adjusted R-squared:  0.2969
F-statistic: 7.195 on 3 and 41 DF,  p-value: 0.0005466

> rikz10 = lm(Richness ~ NAP + angle2 + NAP * angle2, data=rikz)

> summary(rikz10)

Call:
lm(formula = Richness ~ NAP + angle2 + NAP * angle2, data = rikz)

Residuals:
   Min     1Q Median     3Q    Max
-5.608 -2.265 -1.046  1.453 16.257

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.56005    1.41559   2.515   0.0159 *
NAP         -1.81118    1.33015  -1.362   0.1807
angle2       0.05251    0.02131   2.465   0.0180 *
NAP:angle2  -0.01603    0.02081  -0.770   0.4455
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.976 on 41 degrees of freedom
Multiple R-squared:  0.4117,  Adjusted R-squared:  0.3687
F-statistic: 9.565 on 3 and 41 DF,  p-value: 6.513e-05
```

There does appear to be a significant impact on species richness when the interaction effect between angle2 and NAP is considered. However there does not appear to be a significant impact when the interaction effect between angle1 and NAP is considered.

11.11 Model Predictions

The *predict()* function returns predictions from the results of various model fitting functions. Most prediction methods which are similar to those for linear models have an argument *newdata* specifying the first place to look for explanatory variables to be used for prediction.

Input to the *predict()* function must be in the format of a dataframe.

```
> owl = read.csv('owl_data.csv')

> owl.lm = lm(begging~sex+food+sex*food, data = owl)

> newdata = data.frame(sex=rep('Male',20),food=1:20)

> newdata.2 = data.frame(sex=rep('Female',20),food=1:20)

> owl.lm_male = predict(owl.lm, newdata)

> owl.lm_female = predict(owl.lm, newdata.2)

> owl.lm_male
       1        2        3        4        5        6        7        8
28.82591 27.91776 27.00962 26.10147 25.19332 24.28518 23.37703 22.46889
       9       10       11       12       13       14       15       16
21.56074 20.65260 19.74445 18.83630 17.92816 17.02001 16.11187 15.20372
      17       18       19       20
14.29558 13.38743 12.47929 11.57114


> owl_lm_female
       1        2        3        4        5        6        7        8
22.94928 22.23438 21.51949 20.80459 20.08970 19.37480 18.65991 17.94501
       9       10       11       12       13       14       15       16
17.23012 16.51522 15.80033 15.08543 14.37054 13.65564 12.94075 12.22585
      17       18       19       20
11.51096 10.79606 10.08117  9.36627

> colours = c("red","blue", "green",
              "yellow", "purple", "Cyan",
              "pink", "brown"
             )

> title = "Histogram of Owl predictions (Male)"

> title.2 = "Histogram of Owl predictions (Female)"

> hist(owl.lm_male, col = colours, main = title, ylab = "Begging", xlab = "Food")
```

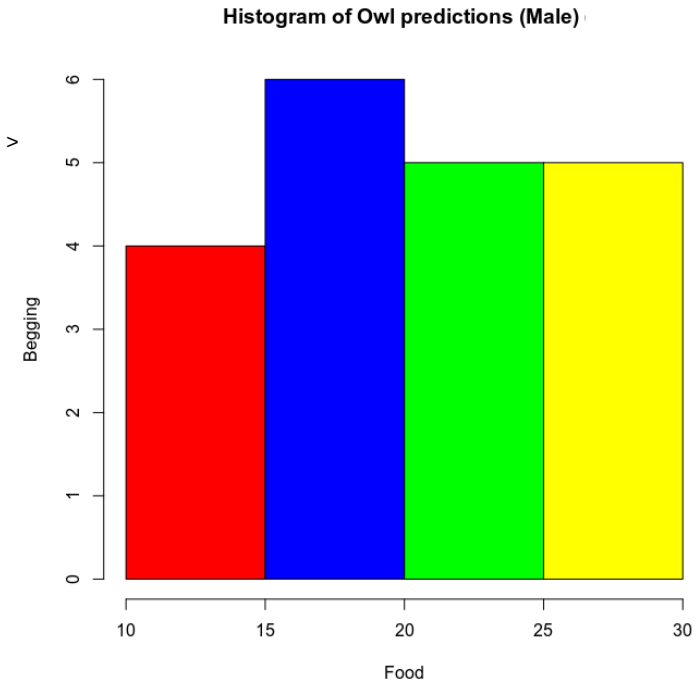


Illustration 15: owl: Predictions (male)

```
hist(owl.lm_female, col = colours, main = title.2, ylab = "Begging", xlab = "Food")
```

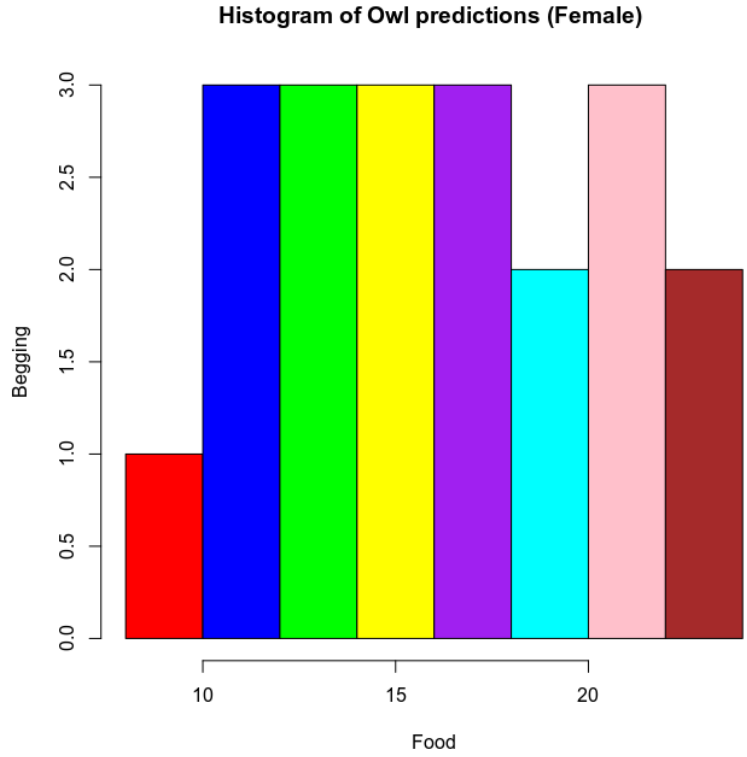


Illustration 16: owl: Predictions (female)

12. Distribution models

Thus far the linear modelling assumed that the residuals fell around the line of regression on a Normal or Gaussian distribution. This form of model is a continuous function which approximates the exact binomial distribution of events. The Gaussian distribution shown is normalised so that the sum over all values of *x* gives a probability of one.

12.1 Standard Deviation

A simple example.

```
> a = c(9, 2, 5, 4, 12, 7, 8, 11, 9, 3, 7, 4, 12, 5, 4, 10, 9, 6, 9, 4)
> a
 [1]  9  2  5  4 12  7  8 11  9  3  7  4 12  5  4 10  9  6  9  4

> b = mean(a)
> b
[1] 7

> c = a-b
> c
 [1]  2 -5 -2 -3  5  0  1  4  2 -4  0 -3  5 -2 -3  3  2 -1  2 -3

> d = c*c
> d
 [1]  4 25  4  9 25  0  1 16  4 16  0  9 25  4  9  9  4  1  4  9

> e = mean(d)
> e
[1] 8.9

> f = sqrt(e)
> f
[1] 2.983287
```

So for the set of data (9, 2, 5, 4, 12, 7, 8, 11, 9, 3, 7, 4, 12, 5, 4, 10, 9, 6, 9, 4) mean (μ) = 7 standard deviation δ = 2.983287

12.1.1 Plotting the Standard Deviation

Note for the plot # - type: the type of plot to be drawn where "n" means do not plot the points

- **xlab:** the title of the x axis
- **ylab:** the title of the y axis
- **main:** the overall title for the plot

- **axes:** when FALSE it suppresses the axis automatically generated by the high level plotting function so that we can create custom axis

```
# Set the sample mean to 7 and SD to 2.8
> sample_mean = 7
> sample_sd = 2.8

# Fill one SD
> sd_to_fill = 1
> lower_bound = sample_mean - sample_sd * sd_to_fill
> upper_bound = sample_mean + sample_sd * sd_to_fill

# Generates equally spaced values within 4 SD of mean
> x = seq(-4, 4, length = 1000) * sample_sd + sample_mean

# The height of the probability distribution at each point
> y = dnorm(x, sample_mean, sample_sd)

# Generate the plot
> plot(x, y, type="n", xlab = "Samples", ylab = "", main = "Distribution of
  Samples", axes = FALSE)

# Connect the points with each other to form a curve
lines(x, y)

# Returns a vector of boolean values to ensure only x values
# between bounds are allowed
> bounds_filter = x >= lower_bound & x <= upper_bound
> x_within_bounds = x[bounds_filter]
> y_within_bounds = y[bounds_filter]

# Bordering the area to be filled
> x_polygon = c(lower_bound, x_within_bounds, upper_bound)
> y_polygon = c(0, y_within_bounds, 0)
> polygon(x_polygon, y_polygon, col = "green")

# Returns the probability that a normally distributed random number
# will be less than the given number
 > probability_within_bounds = pnorm(upper_bound, sample_mean, sample_sd) -
   pnorm(lower_bound, sample_mean, sample_sd)

# Concatenate the various values to display on the curve
  > text = paste("p(", lower_bound, "< height <", upper_bound, ") =",
    signif(probability_within_bounds, digits = 3))

# Display the text on the plot
> mtext(text)>

# Add an axis to the plot
> sd_axis_bounds = 5>
> axis_bounds = seq(-sd_axis_bounds * sample_sd + sample_mean, sd_axis_bounds
  * sample_sd + sample_mean, by = sample_sd)
> axis(side = 1, at = axis_bounds, pos = 0)
```

By changing the value of the *sd_fill* from one to two and three the various areas under the curve for each standard deviation can be seen. There is a simple rule canned the *empirical rule* to remember these. 68.27%, 95.45% and 99.73% of the values lie within one, two and three standard deviations of the mean.

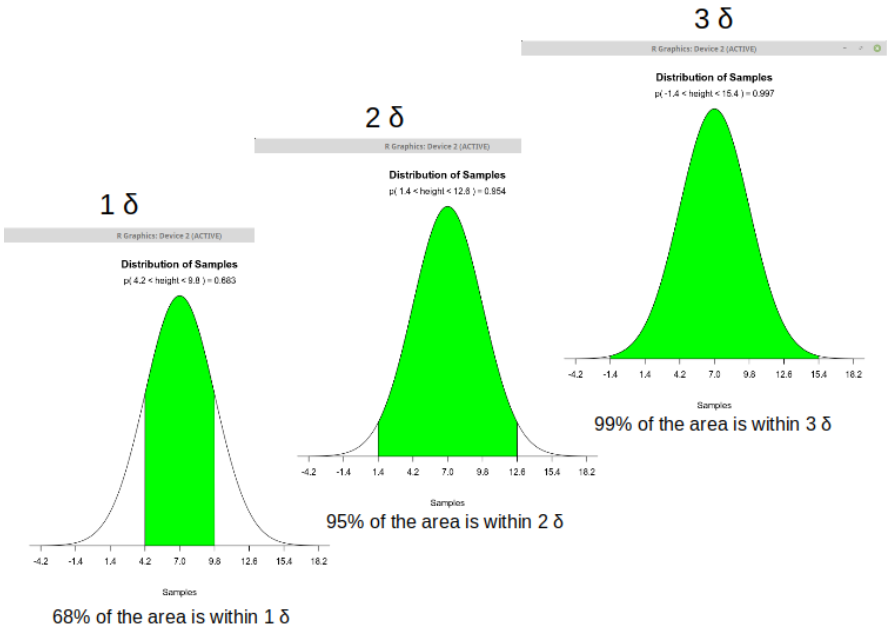


Illustration 17: Standard Deviation

Another example

Looking at another example in more detail.

To calculate the standard deviation of a set of random numbers:

- Work out the mean (μ).
- For each number, subtract the mean (x - μ).
- Square the result $(x - \mu)^2$.
- Get the mean of the squared differences $1/N \Sigma(x - \mu)^2$. This is called the variance (v).
- Get the square root of that $\delta = \sqrt{1/N \Sigma(x - \mu)^2}$.

```
> data.set = round(runif(100,0,1), 2)

> data.set
  [1] 0.56 0.13 0.27 0.91 0.08 0.49 0.99 0.67 0.74 0.59 0.92 0.55 0.18 0.44 0.64
 [16] 0.92 0.15 0.96 0.17 0.67 0.74 0.33 0.52 0.59 0.17 0.59 0.95 0.68 0.71 0.51
 [31] 0.73 0.72 0.71 0.57 0.37 0.73 0.57 0.99 0.84 0.06 0.93 0.16 0.23 0.91 0.75
 [46] 0.89 0.67 0.17 0.29 0.62 0.56 0.35 0.03 0.55 0.62 0.84 0.30 0.69 0.30 0.88
 [61] 0.10 0.03 0.98 0.68 0.38 0.35 0.60 0.82 0.51 0.24 0.58 1.00 0.97 0.65 0.04
 [76] 0.93 0.42 0.97 0.03 0.52 0.38 0.83 0.18 0.37 0.10 0.60 0.96 0.96 0.54 0.65
 [91] 0.75 0.56 0.27 0.94 0.25 0.66 0.41 0.34 0.21 0.99

> mean_ds = mean(data.set)

> mean_ds
[1] 0.556
```

The mean: μ = 0.556

Subtract the Mean from each x to get (x - μ).

```
> sub_ds = data.set - mean_ds
```

```
> sub_ds
  [1]  0.004 -0.426 -0.286  0.354 -0.476 -0.066  0.434  0.114  0.184  0.034
 [11]  0.364 -0.006 -0.376 -0.116  0.084  0.364 -0.406  0.404 -0.386  0.114
 [21]  0.184 -0.226 -0.036  0.034 -0.386  0.034  0.394  0.124  0.154 -0.046
 [31]  0.174  0.164  0.154  0.014 -0.186  0.174  0.014  0.434  0.284 -0.496
 [41]  0.374 -0.396 -0.326  0.354  0.194  0.334  0.114 -0.386 -0.266  0.064
 [51]  0.004 -0.206 -0.526 -0.006  0.064  0.284 -0.256  0.134 -0.256  0.324
 [61] -0.456 -0.526  0.424  0.124 -0.176 -0.206  0.044  0.264 -0.046 -0.316
 [71]  0.024  0.444  0.414  0.094 -0.516  0.374 -0.136  0.414 -0.526 -0.036
 [81] -0.176  0.274 -0.376 -0.186 -0.456  0.044  0.404  0.404 -0.016  0.094
 [91]  0.194  0.004 -0.286  0.384 -0.306  0.104 -0.146 -0.216 -0.346  0.434
```

Square each result to get $(x - \mu)^2$.

```
> sq_ds = sub_ds * sub_ds

> sq_ds
  [1] 0.000016 0.181476 0.081796 0.125316 0.226576 0.004356 0.188356 0.012996
  [9] 0.033856 0.001156 0.132496 0.000036 0.141376 0.013456 0.007056 0.132496
 [17] 0.164836 0.163216 0.148996 0.012996 0.033856 0.051076 0.001296 0.001156
 [25] 0.148996 0.001156 0.155236 0.015376 0.023716 0.002116 0.030276 0.026896
 [33] 0.023716 0.000196 0.034596 0.030276 0.000196 0.188356 0.080656 0.246016
 [41] 0.139876 0.156816 0.106276 0.125316 0.037636 0.111556 0.012996 0.148996
 [49] 0.070756 0.004096 0.000016 0.042436 0.276676 0.000036 0.004096 0.080656
 [57] 0.065536 0.017956 0.065536 0.104976 0.207936 0.276676 0.179776 0.015376
 [65] 0.030976 0.042436 0.001936 0.069696 0.002116 0.099856 0.000576 0.197136
 [73] 0.171396 0.008836 0.266256 0.139876 0.018496 0.171396 0.276676 0.001296
 [81] 0.030976 0.075076 0.141376 0.034596 0.207936 0.001936 0.163216 0.163216
 [89] 0.000256 0.008836 0.037636 0.000016 0.081796 0.147456 0.093636 0.010816
 [97] 0.021316 0.046656 0.119716 0.188356
```

Get the mean of these values $1/N \Sigma(x - \mu)^2$.

```
> mean_ds_sd = mean(sq_ds)

> mean_ds_sd
[1] 0.081938
```

Therefore $\delta = 0.081938$.

12.2 Expand Grid - expand.grid()

Create a data-frame from all combinations of the supplied vectors or factors.

```
> grid.data = expand.grid(height = seq(60,80,5),
                          weight = seq(100, 300, 50),
                          sex = c("Male", "Female")
                         )

> grid.data
   height weight    sex
1      60    100   Male
2      65    100   Male
3      70    100   Male
4      75    100   Male
5      80    100   Male
6      60    150   Male
7      65    150   Male
8      70    150   Male
9      75    150   Male
10     80    150   Male
11     60    200   Male
12     65    200   Male
13     70    200   Male
14     75    200   Male
```

```
15     80    200   Male
16     60    250   Male
17     65    250   Male
18     70    250   Male
19     75    250   Male
20     80    250   Male
21     60    300   Male
22     65    300   Male
23     70    300   Male
24     75    300   Male
25     80    300   Male
26     60    100 Female
27     65    100 Female
28     70    100 Female
29     75    100 Female
30     80    100 Female
31     60    150 Female
32     65    150 Female
33     70    150 Female
34     75    150 Female
35     80    150 Female
36     60    200 Female
37     65    200 Female
38     70    200 Female
39     75    200 Female
40     80    200 Female
41     60    250 Female
42     65    250 Female
43     70    250 Female
44     75    250 Female
45     80    250 Female
46     60    300 Female
47     65    300 Female
48     70    300 Female
49     75    300 Female
50     80    300 Female
```

12.3 Normal or Gaussian distribution

What has been shown is actually a Normal or Gaussian distribution. This is a very common continuous probability distribution that are often used to represent real-valued random variables whose distributions are not known.

12.4 Poisson distribution

The Poisson distribution is the probability distribution of independent event occurrences in an interval. If *lambda (λ)* is the mean occurrence per interval, then the probability of having x occurrences within a given interval is:

$f(x) = \lambda^{x} e^{-\lambda} / x!$ where $x = 0,1,2,3,n$

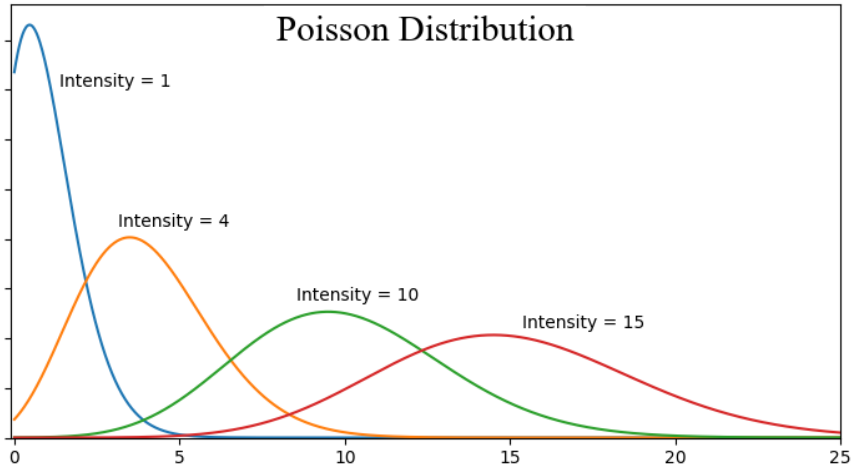


Illustration 18: Poisson distribution

Note that the Poisson distribution is for zero and positive values only. This can be useful if the dataset being operated on cannot have negative values.

For example.

Question: If there are five birds spotted landing on a pond per minute on average, what is the probability of eight birds landing on the pond in any minute?

The probability of having eight or *less* birds landing on the pond in any minute is given by the *ppois()* function. This function is the density, distribution function, quantile function and random generation for the Poisson distribution with parameter *lambda (λ)*. This is called the *lower tail* of the function.

```
> ppois(8, lambda=5)
[1] 0.9319064
```

Secondly it is necessary to calculate the probability of having eight or *more* birds landing on the pond in any minute is called the *upper tail*. Hence the probability of having seventeen or more cars crossing the bridge in a minute is in the upper tail of the probability density function.

```
> ppois(8, lambda=5, lower=FALSE)
[1] 0.06809363
```

So there is a 93% chance of having eight or less birds landing on the pond while there is a 6.8% chance of having eight or more birds landing on the pond in and minute.

12.4.1 Linear Models based on Poisson Distribution

Returning to an earlier example.

```
> setwd('~/datasets/RIKZ_2')

> rikz = read.csv('RIKZ.csv')

> rikz10 = lm(Richness ~ NAP + angle2 + NAP * angle2, data=rikz)
```

```
> summary(rikz10)

Call:
lm(formula = Richness ~ NAP + angle2 + NAP * angle2, data = rikz)

Residuals:
   Min     1Q Median     3Q    Max
-5.608 -2.265 -1.046  1.453 16.257

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.56005    1.41559   2.515   0.0159 *
NAP         -1.81118    1.33015  -1.362   0.1807
angle2       0.05251    0.02131   2.465   0.0180 *
NAP:angle2  -0.01603    0.02081  -0.770   0.4455
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.976 on 41 degrees of freedom
Multiple R-squared:  0.4117,  Adjusted R-squared:  0.3687
F-statistic: 9.565 on 3 and 41 DF,  p-value: 6.513e-05
```

Using the Normal Distribution did not show a p-value of much significance.

Try running a linear model using the Poisson Distribution. In *R* to do this it is necessary to run the *Fitting Generalized Linear Models* which offers the *family* option. to run models based on other distributions like poisson.

```
> rikz11 = glm(Richness ~ NAP + angle2 + NAP * angle2, data=rikz, family=poisson)

> summary(rikz11)

Call:
glm(formula = Richness ~ NAP + angle2 + NAP * angle2, family = poisson,
    data = rikz)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.4543  -1.1122  -0.7233   0.7948   5.0552

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.259472   0.157057   8.019 1.06e-15 ***
NAP         -0.743224   0.188056  -3.952 7.75e-05 ***
angle2       0.008586   0.002176   3.946 7.95e-05 ***
NAP:angle2   0.003209   0.002523   1.272    0.203
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 179.753  on 44  degrees of freedom
Residual deviance:  97.026  on 41  degrees of freedom
AIC: 247.02

Number of Fisher Scoring iterations: 5
```

As can be seen the p-values show much more significance for this distribution model.

12.4.2 Linear Models based on other Distributions

There are many other distribution models. Here is a summary and *R* offers the capability to run tests with each by adjusting the *family* value.

Continuous Data	Data range	Link Function
Normal	Any value, positive or negative	1
Gamma	Positive values only	log
Beta	Values between zero and one	logit
Discrete Data	**Data range**	**Link Function**
Poisson	Zero and positive values	log
Binomial	Zero and positive values	logit

```
> rikz12 = glm(Richness ~ NAP, data=rikz, family=binomial)
Error in eval(family$initialize) : y values must be 0 <= y <= 1
```

In this case //Richness/ would have to be yes/no values.

```
> rikz13 = glm(Richness ~ NAP, data=rikz, family=gamma)
Error in family() : 0 arguments passed to 'gamma' which requires 1
```

In this case *Richness* would have to consist of *count* values.

12.4.3 Link Functions

The link function provides the relationship between the linear predictor and the mean of the distribution function. There are many commonly used link functions, and their choice is informed by several considerations. The table above gives the link functions associated with the various distributions.

12.4.4 Type Response

The type of prediction is required for *predict.glm*. The default is on the scale of the linear predictors; the alternative *response* is on the scale of the response variable. Thus for a default binomial model the default predictions are of log-odds (probabilities on *logit* scale) and *type = "response"* gives the predicted probabilities. The *terms* option returns a matrix giving the fitted values of each term in the model formula on the linear predictor scale.

The alternative type to response is the *type = "terms"* option which returns a matrix giving the fitted values of each term in the model formula on the linear predictor scale.

```
> owl = read.csv('owl_data.csv')

> owl.glm = glm(begging~sex+food+sex*food, family=poisson, data = owl)

> newdata = data.frame(sex=rep('Male',20),food=1:20)

> owl.glm_male = predict.glm(owl.glm, newdata, family=poisson)
```

```
> owl.glm_male
       1        2        3        4        5        6        7        8
5.155740 5.020808 4.885877 4.750945 4.616013 4.481082 4.346150 4.211218
       9       10       11       12       13       14       15       16
4.076286 3.941355 3.806423 3.671491 3.536559 3.401628 3.266696 3.131764
      17       18       19       20
2.996832 2.861901 2.726969 2.592037

> owl.glm_male = predict.glm(owl.glm, newdata, family=poisson, type='response')

> owl.glm_male
        1         2         3         4         5         6         7         8
173.42413 151.53377 132.40650 115.69357 101.09021  88.33015  77.18073  67.43864
        9        10        11        12        13        14        15        16
 58.92623  51.48830  44.98922  39.31048  34.34854  30.01291  26.22455  22.91437
       17        18        19        20
 20.02202  17.49475  15.28648  13.35696

> owl.glm_male = predict.glm(owl.glm, newdata, family=poisson, type='terms')

> owl.glm_male
         sex      food     sex:food
1  0.1482603 3.0438445  0.093027486
2  0.1482603 2.9157238  0.086216467
3  0.1482603 2.7876031  0.079405448
4  0.1482603 2.6594824  0.072594430
5  0.1482603 2.5313617  0.065783411
6  0.1482603 2.4032409  0.058972392
7  0.1482603 2.2751202  0.052161374
8  0.1482603 2.1469995  0.045350355
9  0.1482603 2.0188788  0.038539336
10 0.1482603 1.8907581  0.031728318
11 0.1482603 1.7626373  0.024917299
12 0.1482603 1.6345166  0.018106280
13 0.1482603 1.5063959  0.011295261
14 0.1482603 1.3782752  0.004484243
15 0.1482603 1.2501545 -0.002326776
16 0.1482603 1.1220338 -0.009137795
17 0.1482603 0.9939130 -0.015948813
18 0.1482603 0.8657923 -0.022759832
19 0.1482603 0.7376716 -0.029570851
20 0.1482603 0.6095509 -0.036381869
attr(,"constant")
[1] 1.870608
```

12.5 Exercise: Linear Modelling 1

Using the *insect_spray.csv* data which shows counts of insects after a particular type of insect spray has been used.

1. Run a simple linear model comparing the effectiveness of the different sprays (this will be a traditional ANOVA).
2. Get predictions and their standard errors from this model for each spray type (using the *predict.lm* function).
3. Run the model again as a GLM using the poisson distribution (are the estimates different from model 1? why are they so different?).
4. Get predictions and their standard errors (using the *predict.glm* function). How do the SEs compare to model 1. Why are they so different?
5. Create a matrix to store the predictions and SEs from both models (i.e. four column matrix).
6. Save this matrix table of results as a *.csv* file in your working directory.

Answer:

Import the data.

```
> insect_spray = read.csv('insect_spray.csv')

> names(insect_spray)
[1] "X"     "count" "spray"

> head(insect_spray)
  X count spray
1 1    10     A
2 2     7     A
3 3    20     A
4 4    14     A
5 5    14     A
6 6    12     A

> str(insect_spray)
'data.frame': 72 obs. of  3 variables:
 $ X    : int  1 2 3 4 5 6 7 8 9 10 ...
 $ count: int  10 7 20 14 14 12 10 23 17 20 ...
 $ spray: Factor w/ 6 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1 ...

> summary(insect_spray)
       X             count       spray
 Min.   : 1.00   Min.   : 0.00   A:12
 1st Qu.:18.75   1st Qu.: 3.00   B:12
 Median :36.50   Median : 7.00   C:12
 Mean   :36.50   Mean   : 9.50   D:12
 3rd Qu.:54.25   3rd Qu.:14.25   E:12
 Max.   :72.00   Max.   :26.00   F:12
```

Run a simple linear model comparing the effectiveness of the different sprays.

```
> mod1 = lm(count~spray, data=insect_spray)

> summary(mod1)

Call:
lm(formula = count ~ spray, data = insect_spray)

Residuals:
   Min     1Q Median     3Q    Max
-8.333 -1.958 -0.500  1.667  9.333

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  14.5000     1.1322  12.807  < 2e-16 ***
sprayB        0.8333     1.6011   0.520    0.604
sprayC      -12.4167     1.6011  -7.755 7.27e-11 ***
sprayD       -9.5833     1.6011  -5.985 9.82e-08 ***
sprayE      -11.0000     1.6011  -6.870 2.75e-09 ***
sprayF        2.1667     1.6011   1.353    0.181
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.922 on 66 degrees of freedom
Multiple R-squared:  0.7244,  Adjusted R-squared:  0.7036
F-statistic:  34.7 on 5 and 66 DF,  p-value: < 2.2e-16
```

```
> anova(mod1)
Analysis of Variance Table

Response: count
          Df Sum Sq Mean Sq F value    Pr(>F)
spray      5 2668.8  533.77  34.702 < 2.2e-16 ***
Residuals 66 1015.2   15.38
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Get predictions and their standard errors from this model for each spray type.

```
> new.data = data.frame(spray=c("A", "B", "C", "D", "E", "F"))

> pred1 = predict.lm(mod1, new.data, se.fit=T)

> summary(pred1)
              Length Class  Mode
fit           6      -none- numeric
se.fit        6      -none- numeric
df            1      -none- numeric
residual.scale 1     -none- numeric

> pred1
$fit
        1         2         3         4         5         6
14.500000 15.333333  2.083333  4.916667  3.500000 16.666667

$se.fit
       1        2        3        4        5        6
1.132156 1.132156 1.132156 1.132156 1.132156 1.132156

$df
[1] 66

$residual.scale
[1] 3.921902
```

Run the model again as a GLM using the poisson distribution. Are the estimates different from model one?

```
> mod2 = glm(count~spray, data=insect_spray, family=poisson)

> summary(mod2)

Call:
glm(formula = count ~ spray, family = poisson, data = insect_spray)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.3852  -0.8876  -0.1482   0.6063   2.6922

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  2.67415    0.07581  35.274  < 2e-16 ***
sprayB       0.05588    0.10574   0.528    0.597
sprayC      -1.94018    0.21389  -9.071  < 2e-16 ***
sprayD      -1.08152    0.15065  -7.179 7.03e-13 ***
sprayE      -1.42139    0.17192  -8.268  < 2e-16 ***
sprayF       0.13926    0.10367   1.343    0.179
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 409.041  on 71  degrees of freedom
Residual deviance:  98.329  on 66  degrees of freedom
```

```
AIC: 376.59

Number of Fisher Scoring iterations: 5

> anova(mod2)
Analysis of Deviance Table

Model: poisson, link: log

Response: count

Terms added sequentially (first to last)


      Df Deviance Resid. Df Resid. Dev
NULL                     71     409.04
spray  5   310.71        66      98.33
```

Get predictions and their standard errors (using the *predict.glm* function). How do the SEs compare to model one. Why are they so different?

```
> new.data = data.frame(spray=c("A", "B", "C", "D", "E", "F"))

> pred2 = predict.glm(mod2, new.data, se.fit=T, type="response")

> summary(pred2)
              Length Class  Mode
fit           6      -none- numeric
se.fit        6      -none- numeric
residual.scale 1      -none- numeric

> pred2
$fit
       1         2         3         4         5         6
14.500000 15.333333  2.083333  4.916667  3.500000 16.666667

$se.fit
       1         2         3         4         5         6
1.0992422 1.1303883 0.4166664 0.6400955 0.5400617 1.1785113

$residual.scale
[1] 1
```

Save this matrix table of results as a *.csv* file in your working directory.

```
> store = matrix(0, nrow=6, ncol=4)
> store[,1] = pred1$fit
> store[,2] = pred1$se.fit
> store[,3] = pred2$fit
> store[,4] = pred2$se.fit

> write.csv(store, "insect_pred_table.csv")

> quit()
Save workspace image? [y/n/c]: n

$ cat insect_pred_table.csv
"","V1","V2","V3","V4"
"1",14.5,1.13215550799177,14.5,1.0992421631893
"2",15.3333333333333,1.13215550799177,15.3333333333333,1.13038833052086
"3",2.08333333333331,1.13215550799177,2.08333333333498,0.416666404601133
"4",4.91666666666668,1.13215550799177,4.91666666666666,0.640095478972524
"5",3.50000000000001,1.13215550799177,3.49999999999999,0.540061724814324
"6",16.6666666666667,1.13215550799177,16.6666666666667,1.17851130197669
```

12.6 Exercise: Linear Modelling 2

Using the *CO2.csv* data which shows the uptake of carbon dioxide of two plant types when kept at different temperatures and at different concentrations of CO_2.

1. Run a simple linear model (regression) comparing the uptake of CO_2 to the concentration of CO_2.
2. Run a simple linear model (t-test) comparing the effect of temperature on CO_2 uptake.
3. Combine these two explanatory variables into a model that includes them both, and also look at one that includes an interaction between them.
4. Get predictions for the two types of plants, for a range of CO_2 concentrations.
5. Rerun the models and get predictions for a Gamma distribution.
6. Compare the different models and find the best model by looking at their Akaike Information Criterion (AIC).

Note: The AIC is an estimator of the relative quality of statistical models for a given set of data. Given a collection of models for the data, AIC estimates the quality of each model, relative to each of the other models. Thus, AIC provides a means for model selection. The AIC can be used to help decide which model is better at describing your data and making predictions from it. The lower the AIC, the better the model.

HINT: an example of using the function is *AIC(mod1, mod2, mod3)*.

Answer:

Import the data.

```
> co2 = read.csv('CO2.csv')

> names(co2)
[1] "X"         "Plant"     "Type"      "Treatment" "conc"      "uptake"

> head(co2)
  X Plant   Type  Treatment conc uptake
1 1   Qn1 Quebec nonchilled   95   16.0
2 2   Qn1 Quebec nonchilled  175   30.4
3 3   Qn1 Quebec nonchilled  250   34.8
4 4   Qn1 Quebec nonchilled  350   37.2
5 5   Qn1 Quebec nonchilled  500   35.3
6 6   Qn1 Quebec nonchilled  675   39.2

> str(co2)
'data.frame': 84 obs. of  6 variables:
 $ X        : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Plant    : Factor w/ 12 levels "Mc1","Mc2","Mc3",..: 10 10 10 10 10 10 10 11 11
11 ...
 $ Type     : Factor w/ 2 levels "Mississippi",..: 2 2 2 2 2 2 2 2 2 ...
 $ Treatment: Factor w/ 2 levels "chilled","nonchilled": 2 2 2 2 2 2 2 2 2 ...
 $ conc     : int  95 175 250 350 500 675 1000 95 175 250 ...
 $ uptake   : num  16 30.4 34.8 37.2 35.3 39.2 39.7 13.6 27.3 37.1 ...
```

```
> summary(co2)
       X             Plant             Type         Treatment       conc
 Min.   : 1.00   Mc1    : 7   Mississippi:42   chilled   :42   Min.   :  95
 1st Qu.:21.75   Mc2    : 7   Quebec     :42   nonchilled:42   1st Qu.: 175
 Median :42.50   Mc3    : 7                                    Median : 350
 Mean   :42.50   Mn1    : 7                                    Mean   : 435
 3rd Qu.:63.25   Mn2    : 7                                    3rd Qu.: 675
 Max.   :84.00   Mn3    : 7                                    Max.   :1000
                 (Other):42
     uptake
 Min.   : 7.70
 1st Qu.:17.90
 Median :28.30
 Mean   :27.21
 3rd Qu.:37.12
 Max.   :45.50
```

Run a simple linear model (regression) comparing the uptake of CO_2 to the concentration of CO_2.

```
> mod1 = lm(uptake~conc, data=co2)

> summary(mod1)

Call:
lm(formula = uptake ~ conc, data = co2)

Residuals:
    Min      1Q  Median      3Q     Max
-22.831  -7.729   1.483   7.748  16.394

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 19.500290   1.853080  10.523  < 2e-16 ***
conc         0.017731   0.003529   5.024 2.91e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.514 on 82 degrees of freedom
Multiple R-squared:  0.2354,  Adjusted R-squared:  0.2261
F-statistic: 25.25 on 1 and 82 DF,  p-value: 2.906e-06
```

Run a simple linear model (t-test) comparing the effect of temperature on CO_2 uptake.

```
> mod2 = lm(uptake~Treatment, data=co2)

> summary(mod2)

Call:
lm(formula = uptake ~ Treatment, data = co2)

Residuals:
     Min       1Q   Median       3Q      Max
-20.0429  -8.6530  -0.4429   9.7321  18.6167

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)          23.783      1.591  14.948   <2e-16 ***
Treatmentnonchilled   6.860      2.250   3.048   0.0031 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.31 on 82 degrees of freedom
Multiple R-squared:  0.1018,  Adjusted R-squared:  0.09084
F-statistic: 9.293 on 1 and 82 DF,  p-value: 0.003096
```

Combine these two explanatory variables into a model that includes them both, and also look at one that includes an interaction between them.

```
> mod3a = lm(uptake~conc + Treatment, data=co2)

> mod3b = lm(uptake~conc * Treatment, data=co2)

> summary(mod3a)

Call:
lm(formula = uptake ~ conc + Treatment, data = co2)

Residuals:
    Min      1Q  Median      3Q     Max
-19.401  -7.066  -1.168   7.573  17.597

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)         16.070528   1.989746   8.077 5.31e-12 ***
conc                 0.017731   0.003306   5.364 7.55e-07 ***
Treatmentnonchilled  6.859524   1.944840   3.527 0.000695 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8.912 on 81 degrees of freedom
Multiple R-squared:  0.3372,  Adjusted R-squared:  0.3208
F-statistic:  20.6 on 2 and 81 DF,  p-value: 5.837e-08

> summary(mod3b)

Call:
lm(formula = uptake ~ conc * Treatment, data = co2)

Residuals:
    Min      1Q  Median      3Q     Max
-18.218  -7.401  -1.117   7.835  17.209

Coefficients:
                         Estimate Std. Error t value Pr(>|t|)
(Intercept)             16.981416   2.464160   6.891 1.15e-09 ***
conc                     0.015637   0.004693   3.332  0.00131 **
Treatmentnonchilled      5.037747   3.484848   1.446  0.15219
conc:Treatmentnonchilled 0.004188   0.006636   0.631  0.52979
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8.946 on 80 degrees of freedom
Multiple R-squared:  0.3405,  Adjusted R-squared:  0.3157
F-statistic: 13.77 on 3 and 80 DF,  p-value: 2.528e-07
```

Get predictions for the two types of plants, for a range of CO_2 concentrations.

```
> mod4 = lm(uptake~conc * Type, data=co2)

> new.data = expand.grid(Type=c("Quebec", "Mississippi"),
                         conc=seq(100,1000,100)
                         )
```

```
> predict.lm(mod4, new.data, se.fit=T)
$fit
       1        2        3        4        5        6        7        8
25.81104 16.73566 28.11905 17.97377 30.42705 19.21188 32.73506 20.44999
       9       10       11       12       13       14       15       16
35.04306 21.68811 37.35106 22.92622 39.65907 24.16433 41.96707 25.40245
      17       18       19       20
44.27508 26.64056 46.58308 27.87867

$se.fit
       1        2        3        4        5        6        7        8
1.622000 1.622000 1.369811 1.369811 1.177546 1.177546 1.077770 1.077770
       9       10       11       12       13       14       15       16
1.096037 1.096037 1.227089 1.227089 1.440463 1.440463 1.705538 1.705538
      17       18       19       20
2.001880 2.001880 2.317526 2.317526

$df
[1] 80

$residual.scale
[1] 6.935822
```

Rerun the models and get predictions for a Gamma distribution.

```
> mod5 = glm(uptake~conc * Type, data=co2, family=Gamma)

> summary(mod5)

Call:
glm(formula = uptake ~ conc * Type, family = Gamma, data = co2)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-0.90639  -0.22457  -0.01455   0.19478   0.54514

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)      6.020e-02  4.361e-03  13.803  < 2e-16 ***
conc            -2.532e-05  6.898e-06  -3.671 0.000434 ***
TypeQuebec      -2.148e-02  5.156e-03  -4.166 7.78e-05 ***
conc:TypeQuebec  7.311e-06  8.094e-06   0.903 0.369081
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for Gamma family taken to be 0.0939836)

    Null deviance: 15.8858  on 83  degrees of freedom
Residual deviance:  8.4588  on 80  degrees of freedom
AIC: 596.1

Number of Fisher Scoring iterations: 5

> new.data = expand.grid(Type=c("Quebec", "Mississippi"),
                         conc=seq(100,1000,100)
                        )
```

```
> predict.glm(mod5, new.data, se.fit=T, type="response")
$fit
       1        2        3        4        5        6        7        8
27.08678 17.34099 28.47595 18.13741 30.01533 19.01050 31.73064 19.97190
       9       10       11       12       13       14       15       16
33.65390 21.03572 35.82534 22.21925 38.29633 23.54390 41.13343 25.03650
      17       18       19       20
44.42452 26.73116 48.28806 28.67189

$se.fit
        1         2         3         4         5         6         7         8
1.7561008 1.1382195 1.6721912 1.0702451 1.5951945 1.0088967 1.5520705 0.9732649
        9        10        11        12        13        14        15        16
1.5921838 0.9959336 1.7853024 1.1180932 2.2010333 1.3739928 2.8953715 1.7851440
       17        18        19        20
3.9288835 2.3729936 5.3967264 3.1730722

$residual.scale
[1] 0.3065674
```

Compare the different models and find the best model by looking at their AIC.

```
> AIC(mod1, mod2, mod3a, mod3b, mod4, mod5)
      df      AIC
mod1   3 620.8180
mod2   3 634.3456
mod3a  4 610.8169
mod3b  5 612.3997
mod4   5 569.6488
mod5   5 596.0955
```

13. Plots

Simple plot. *plot()* is a generic base graphic function for the plotting of *R* objects. *x* and *y* objects must be given to the function.

```
> x = 1:5

> y = 1:5

> plot(x,y)
```

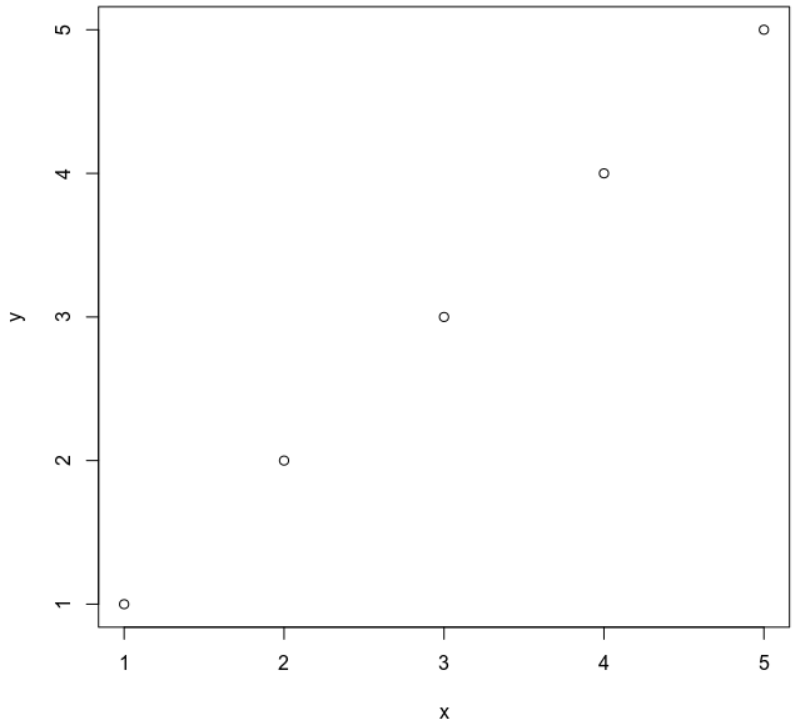


Illustration 19: Simple plot

```
plot(x=c(1,2,3,4), y=c(1,2,3,4))
```

```
> args(plot)
function (x, y, ...)
NULL
```

13.1 Beautify the plot

Add type to adjust the view of the plot. What type of plot should be drawn. Possible types are:

p	points
l	lines
b	both
c	the lines part alone of **b**
o	both over|plotted
h	histogram like (or high|density) vertical lines
s	stair steps
n	no plotting

```
> x = 1:5

> y = 1:5

> plot(x,y, type = "b")
```

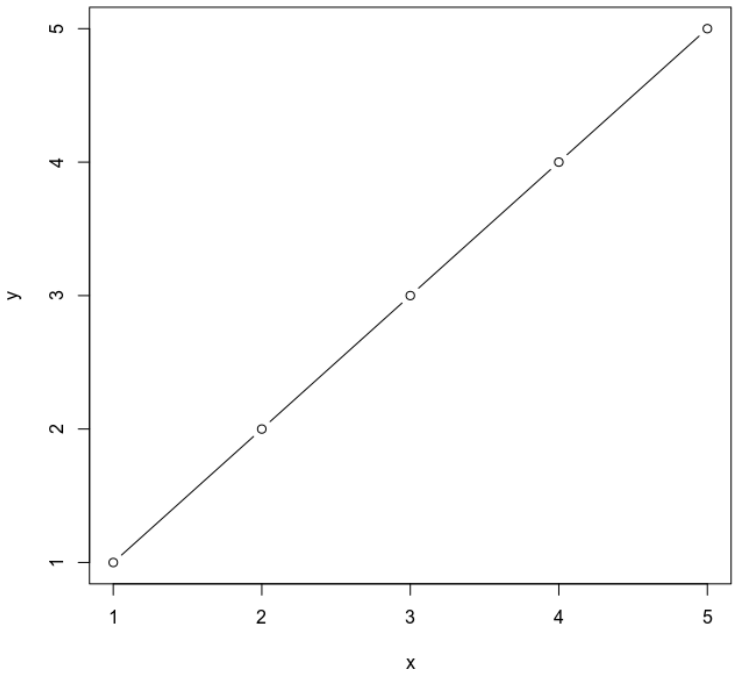


Illustration 20: Plot with lines

Add some colour.

```
> plot(x,y, type = "b", col = "green")
```

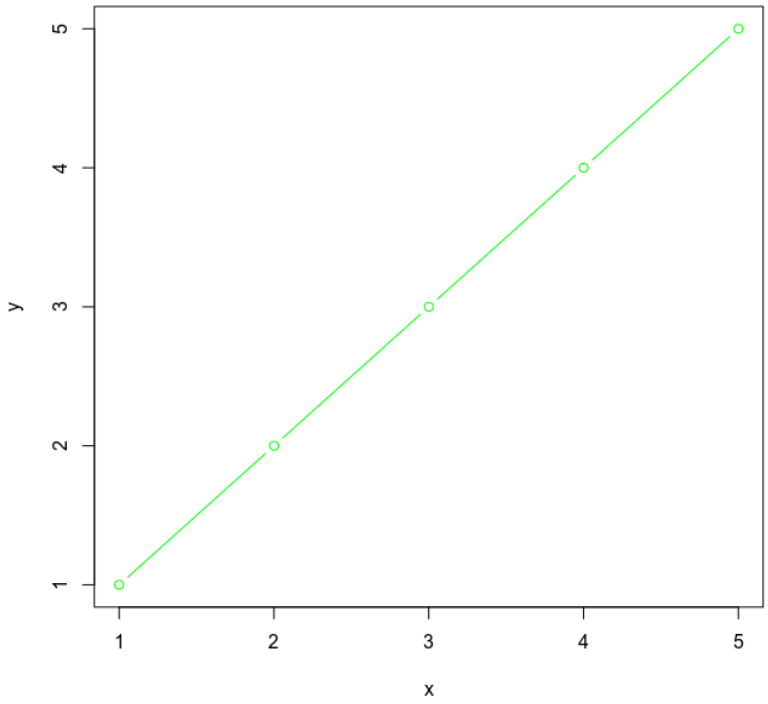


Illustration 21: Plot with colour

It also works as a general plot function for many object types. See Illustration 22 for a simple linear regression.

```
> x = c(1,2,2,3,2,3,4,3,4,5,3)

> y = c(4,8,6,3,5,7,9,2,1,7,4)

> model.1 = lm(y~x)

> class(model.1)
[1] "lm"

> plot(model.1)
```

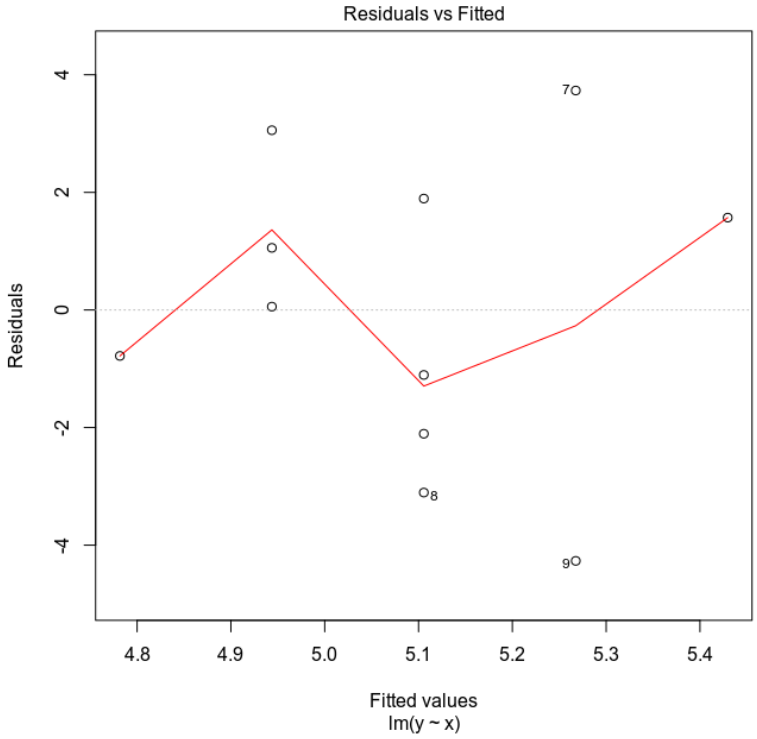


Illustration 22: Plot - model1

13.1.1 Plot layers

Add some other values.

```
> plot(x,y, type = "b", col = "green",xlab = "bottom", ylab = "left")

> title(main = "My plot", xlab = "length", ylab = "height")
```

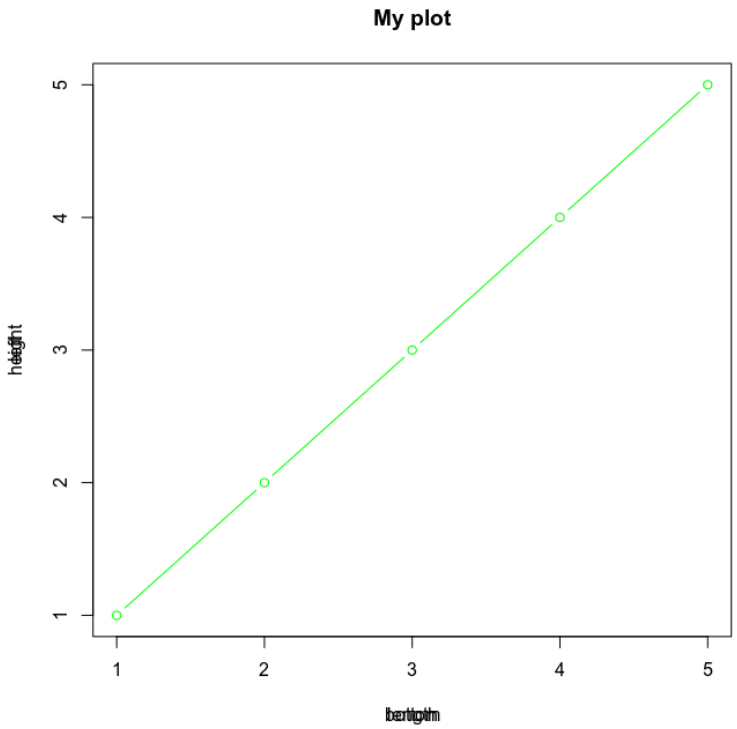


Illustration 23: Plot layers

Notice how each or these lines layered on to of each-other. It is necessary to turn off some things in the first line.

```
> plot(x,y, type = "b", col = "green",xlab = "", ylab = "", xaxt="n", xaxt="n")

> title(main = "My plot", xlab = "length", ylab = "height")
```

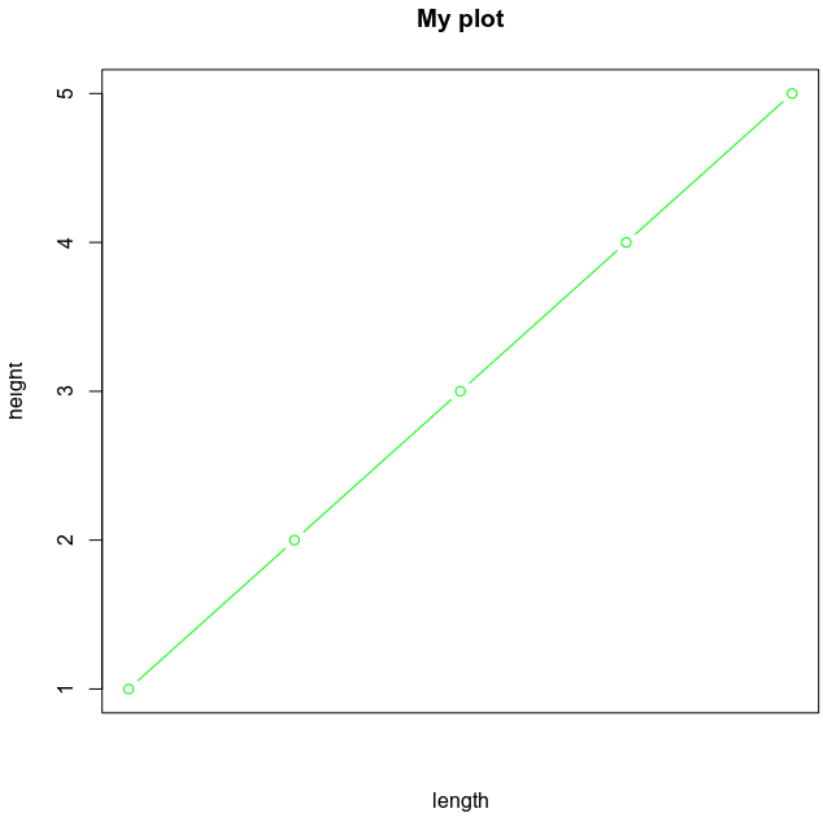


Illustration 24: Plot layers 2

13.1.2 Box type

Use the argument *box type* [bty] to adjust the box.

- **o:** The default value draws a complete rectangle around the plot.
- **n:** Draws nothing around the plot.
- **l**, **7**, **c**, **u**, or **]**: Draws a shape around the plot area that resembles the uppercase letter of the option. So, the option bty="l" draws a line to the left and bottom of the plot.

```
> plot(rnorm(100))
> plot(rnorm(100),bty="o")
> plot(rnorm(100),bty="l")
> plot(rnorm(100),bty="7")
> plot(rnorm(100),bty="c")
> plot(rnorm(100),bty="]")
> plot(rnorm(100),bty="n")
```

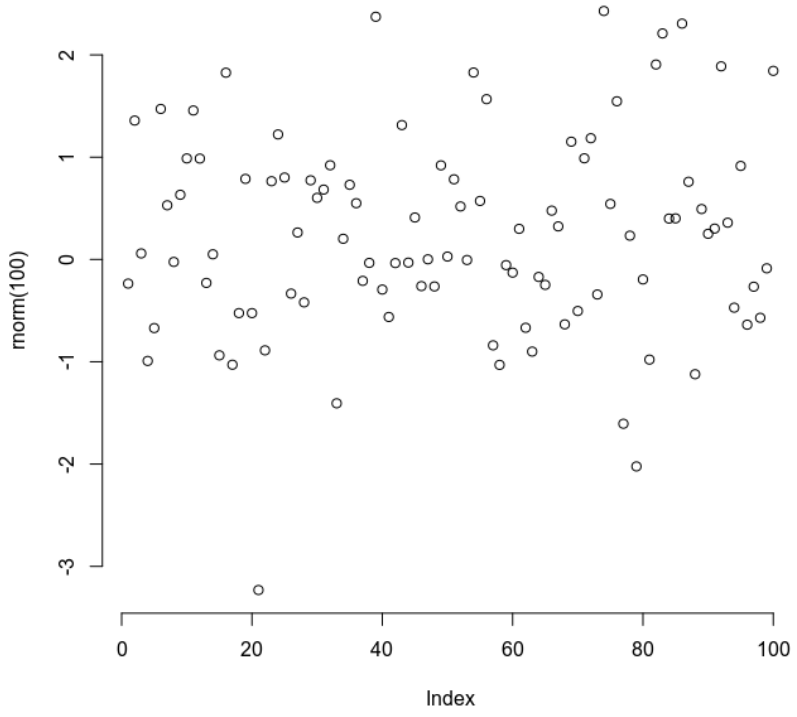


Illustration 25: Box type

13.1.3 Splitting arguments

```
> plot(x,y, xlab="height", ylab="length", cex.axis= 1.2,
          cex.lab=1.5, typ="p", col="red", bty="l", pch=16,
          tck=0.03
         )

> plot(x,y,                          # call the plot
       xlab="height", ylab="length", # label the axes
       xlim=c(0,10),                  # set specific limits to x-axis
       cex.axis= 1.2, cex.lab=1.5,    # set character size for axis & labels
       typ="p", col="red",            # set plot type & colour
       bty="l",                       # set box type around plot
       pch=16,                        # set point character type
       tck=0.03                       # set axis tick marks (+ve is inside plot)
      )
```

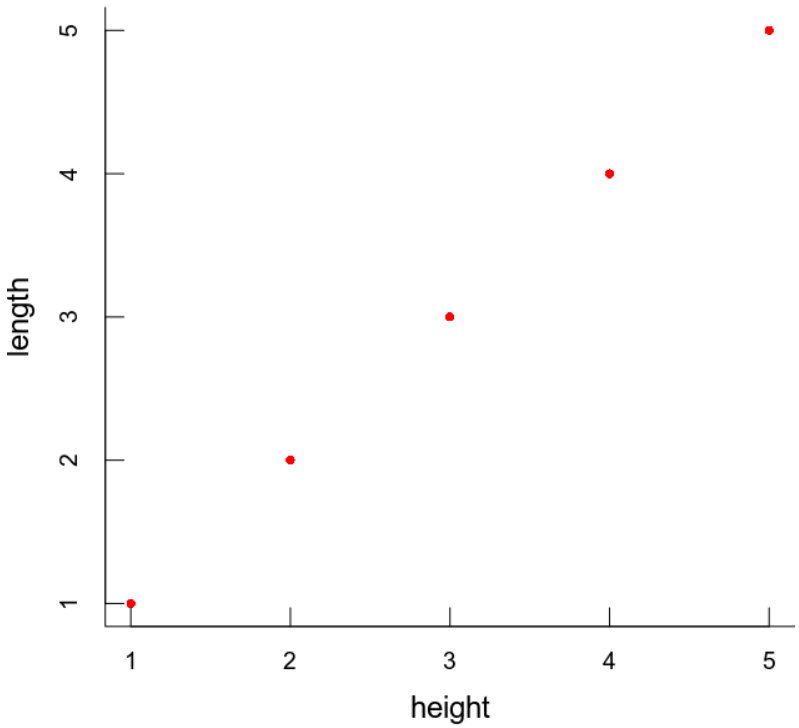


Illustration 26: Plot - splitting arguments

13.1.4 Graphical parameters: par

par(): can be used to set or query graphical parameters. Parameters can be set by specifying them as arguments to *par* in *tag = value* form, or by passing them as a list of tagged values. *par()* on its own returns the current settings for default graphical parameters. These defaults can be modified in *par()*. Some graphical parameters must be set in *par()* like background colour.

```
> par(lty=2,          # Set the line type
      pch=17,         # Define the plotting symbol
      cex.axis=3      # Specify the size of the tick labels
     )
```

Icon type: PCH

Sets the icon type.

pch = 0	square	pch = 13	circle cross
pch = 1	circle	pch = 14	square and triangle down
pch = 2	triangle point up	pch = 15	filled square
pch = 3	plus	pch = 16	filled circle
pch = 4	cross	pch = 17	filled triangle point up
pch = 5	diamond	pch = 18	filled diamond
pch = 6	triangle point down	pch = 19	solid circle up
pch = 7	square cross	pch = 20	bullet (smaller circle)
pch = 8	star	pch = 21	filled circle blue
pch = 9	diamond plus	pch = 22	filled square blue
pch = 10	circle plus	pch = 23	filled diamond blue
pch = 11	triangles up and down	pch = 24	filled triangle point up blue
pch = 12	square plus	pch = 25	filled triangle point down blue

13.1.5 Colours

R has 657 colours to choose from. The colours() function gives a list of the available colour names.

colours(): Creates a list of all available colours.

```
colours = colours()
> head(colours)
[1] "white"         "aliceblue"     "antiquewhite"  "antiquewhite1"
[5] "antiquewhite2" "antiquewhite3"
```

rainbow(n): Creates a vector of *n* contiguous colors.

```
> rainbow = rainbow(5)

> rainbow
[1] "#FF0000FF" "#CCFF00FF" "#00FF66FF" "#0066FFFF" "#CC00FFFF"
```

grey(level): Creates a vector of colours from a vector of gray levels. These are given as a vector of desired gray levels between zero and one; zero indicates *black* and one indicates *white*.

```
> grey = grey(c(0.1,0.3,0.5,0.7,0.9))

> grey
[1] "#1A1A1A" "#4D4D4D" "#808080" "#B3B3B3" "#E6E6E6"
```

13.1.6 text

text(): Insert text at any position of a current open plot.

```
> text(x, y,     # x,y plot co-ordinates for the text
       labels,   # Text to be inserted
       pos,      # Position (1,2,3,4=below,left,above,right)
       offset,   # Distance of pos offset from x,y
       col, cex, font ...  # Other options
      )
```

Example:

```
> plot(1,1)

> text(1,1.05,
       "This is a dot in the middle",
       col="red",cex=0.8
      )

> text(1,1,
       "This is the test to be printed",
       pos=1,offset=1,col="red",cex=0.8
      )
```

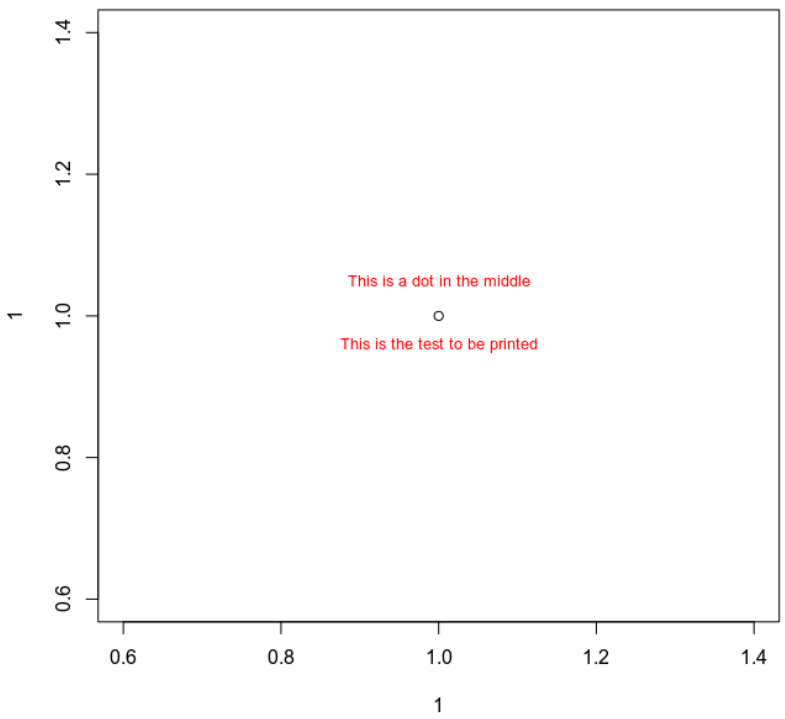


Illustration 27: Plot text

Another example.

```
> plot(0,1,xaxt='n',yaxt='n')
> my_text = list(bquote( paste( "Average Area=5.78" , m^2/h) ),
                 bquote( paste( "Average Distance=12.2", km/h ) )
                )
>
> mtext(side=1,do.call(expression, my_text), line=-1:-2, adj=0)
```

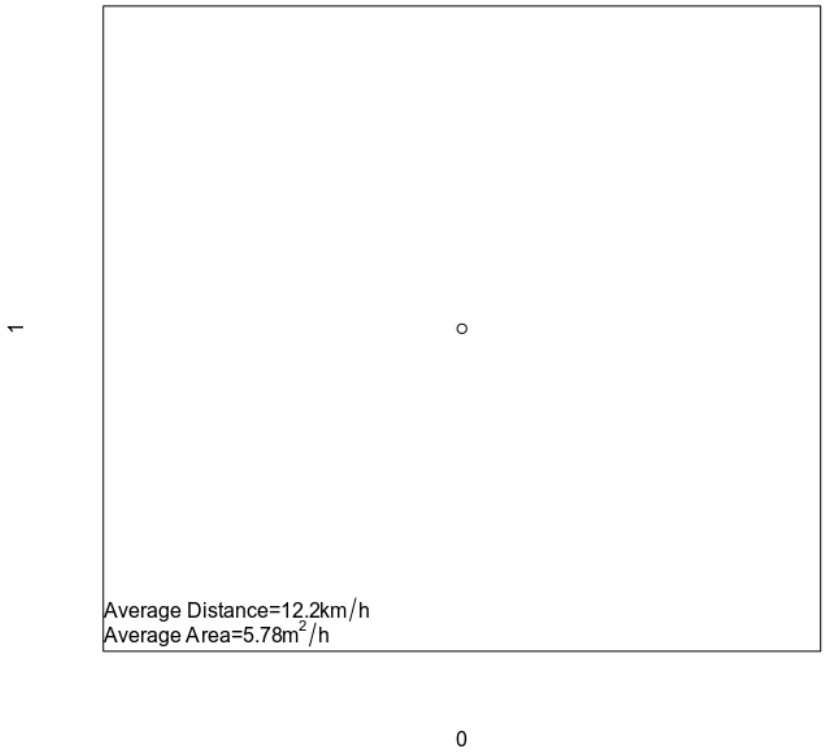


Illustration 28: Plot text 2

13.1.7 Points

points(): Insert points at any position of a current open plot.

```
points(x, y,
       pch,     # Point character type
       cex,     # Character expansion - i.e size
       col,     # Colour of the point
       bg       # Background colour or point
      )
```

Example:

```
> plot(c(0,2),c(0,25), type='n')

> points(x=rep(1,25), y=1:25, pch=1:25)
```

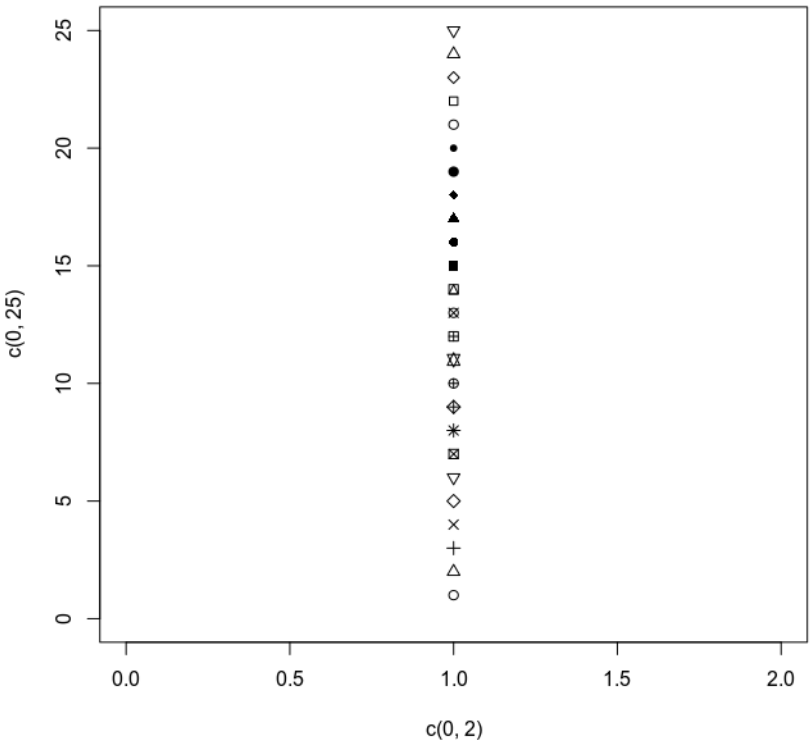


Illustration 29: Plot - points

13.1.8 Symbols

symbols(): Insert a shape at any position of a currently open plot.

```
symbols(x, y, # x,y plot co-ordinates
        circles,        # Draw circle with of diameter (e.g. circle=0.8)
        squares,        # Draw square of side length (e.g. square=0.5)
        rectangles,     # Draw rectangle (side lengths specified by matrix)
        stars,          # Draw star (points etc specified by 3 column matrix)
        thermometers,   # Craw with matrix of 3 or 4 columns
        boxplots,       # Draw with matrix of 5 columns
        add=T,          # Adds the symbol to the current plot
        bg, fg, lwd...  # General settings - see help(symbols)
       )
```

Example:

```
> plot(c(0,4),c(0,6), type='n')

> symbols(x=1, y=3, circles=3,
          bg='red', fg='green',
          lwd=3, add=T
         )
```

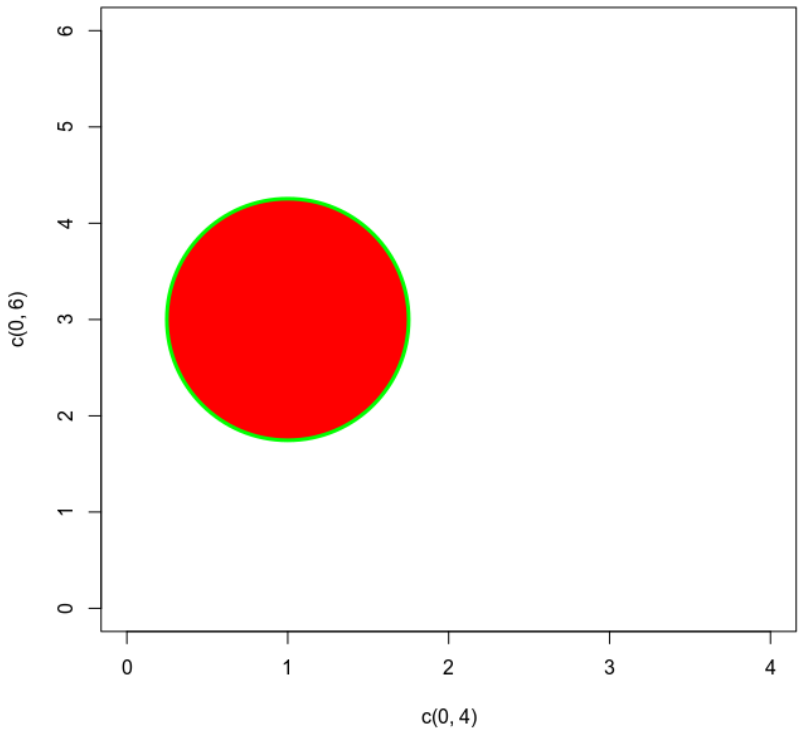


Illustration 30: Plot - symbols

13.1.9 Segments

segments(): Insert line segment at any position of a currently open plot.

```
> segments(x0, y0,    # x,y plot co-ordinates for the start of the segment
           x1, y1,    # x,y plot co-ordinates for the end of the segment
           col,       # Colour of the line = 1:8
           lty,       # Line type = 1:6 (dashed, dotted, whole etc)
           lwd        # Line width
          )
```

Example:

```
> plot(c(0,4),c(0,6), type='n')

> segments(x0=rep(1,6), y0=1:6,
           x1=rep(3,6), y1=1:6,
           lty=1:25
          )
```

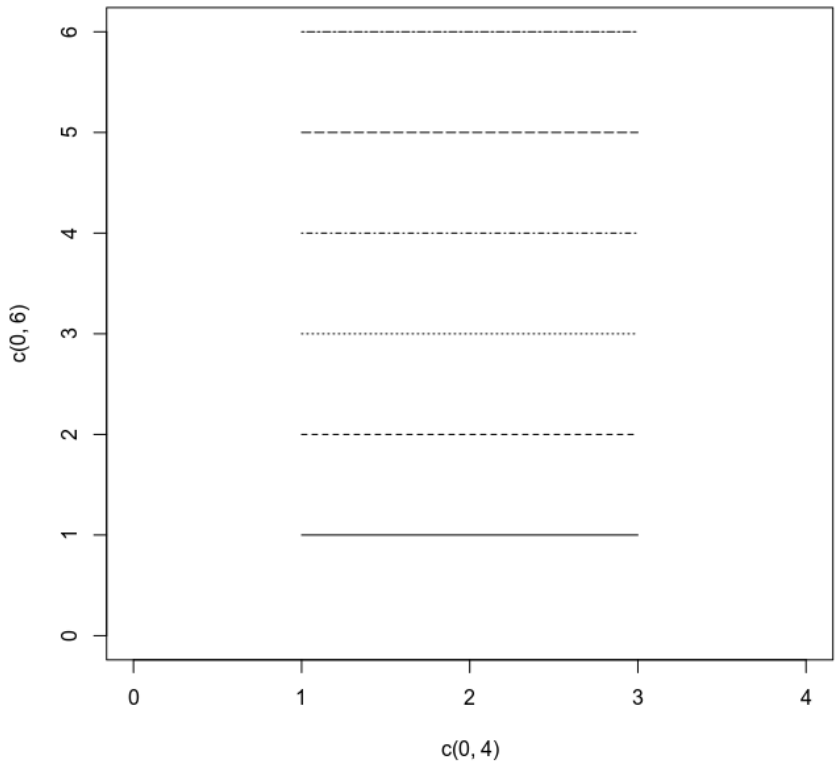


Illustration 31: Plot - lines

13.1.10 Polygon

polygon(): Insert any shape at any position of a currently open plot.

```
  > polygon(x,        # Vector of x co-ordinates for each point to be joined to make
the shape
            y,        # Vector of corresponding y co-ordinates for the shape (matched
to x)
           density, # Density of shading lines that fill the object
           angle,   # Slope of shading lines that fill the object
           col,     # Colour of shading lines (if density=NA then col is fill colour)
           border,  # Colour of the border
           lwd...   # Line width
          )
```

Example:

```
> plot(c(0,8),c(0,3), type='n')

> polygon(c(1,1:7,7), c(0,1,2,1,2,1,2,1,0), col='blue', lwd=3)

> polygon(c(1,7,7), c(0,0,1), col='yellow', angle=90, density=7)
```

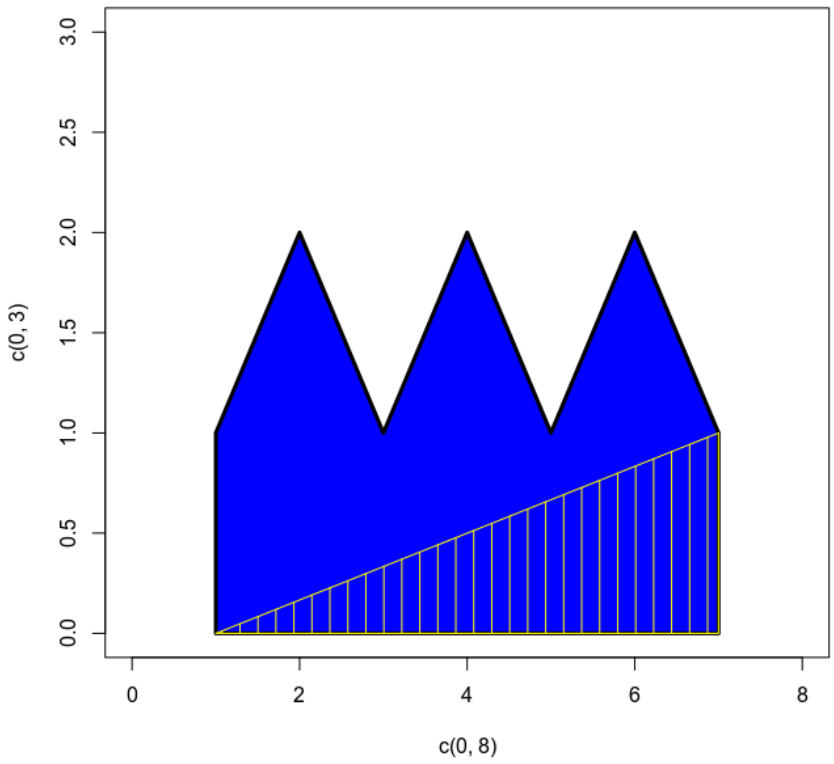


Illustration 32: Plot - polygons

13.1.11 Arrows

arrows(): Draw an arrow between pairs of coordinates in a current open plot.

```
> arrows(x0, y0,          # Co-ordinates of points from which to draw
         x1, y1,          # Co-ordinates of points to which to draw
         length,          # Length of edge of arrow head
         angle,           # Angle from arrow shaft to arrow head
         code,            # 1:3 specifying the type of arrow head
         col, lty, lwd... # Other options
        )
```

Example:

```
> plot(c(0,8),c(0,3), type='n')

> arrows(x0=1,y0=1,x1=6,y1=2,col='dark green',lty=2, lwd=3)
```

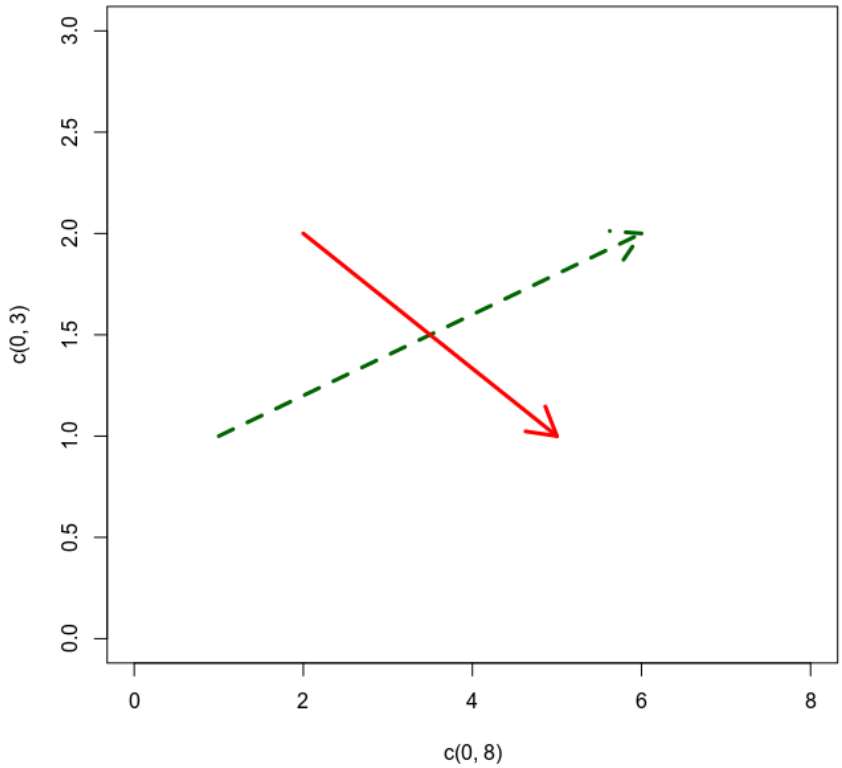


Illustration 33: Plot - arrows

13.1.12 Lines

- **lines():** A generic function taking coordinates given in various ways and joining the corresponding points with line segments.
- **abline():** This function adds one or more straight lines through the current plot. It can specify intercept and slope, horizontal or vertical and can take intercept & slope from an *lm* object.
- **curve()** Draws a curve corresponding to a function over the interval *from, to*. *curve()* can plot also an expression in the variable *xname*, default *x*. Can add the line of an equation to a plot. Note: must set *add=TRUE*.

Example:

```
> plot(c(0,8),c(0,8), type='n')
> lines(x=c(1,5,8), y=c(2,3,1), col='blue', lwd=3)
> x=c(1,2,3,4,5); y=c(2,2,1,1,0)
> abline(lm(y~x))
> curve(x+0.1*x^2, add=T)
```

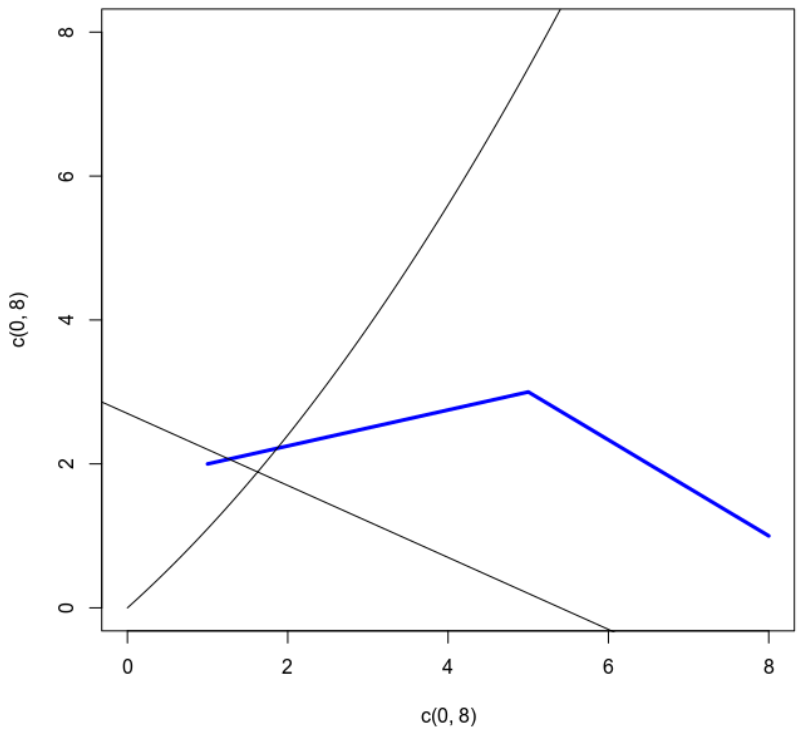


Illustration 34: Plot - lines, curves

Example: abline()

```
> plot(1:21,-10:10)

> abline(h=0, lty=2)      # Put horizontal line at y=0

> abline(v=5, lty=1)      # Put vertical line at x=5
```

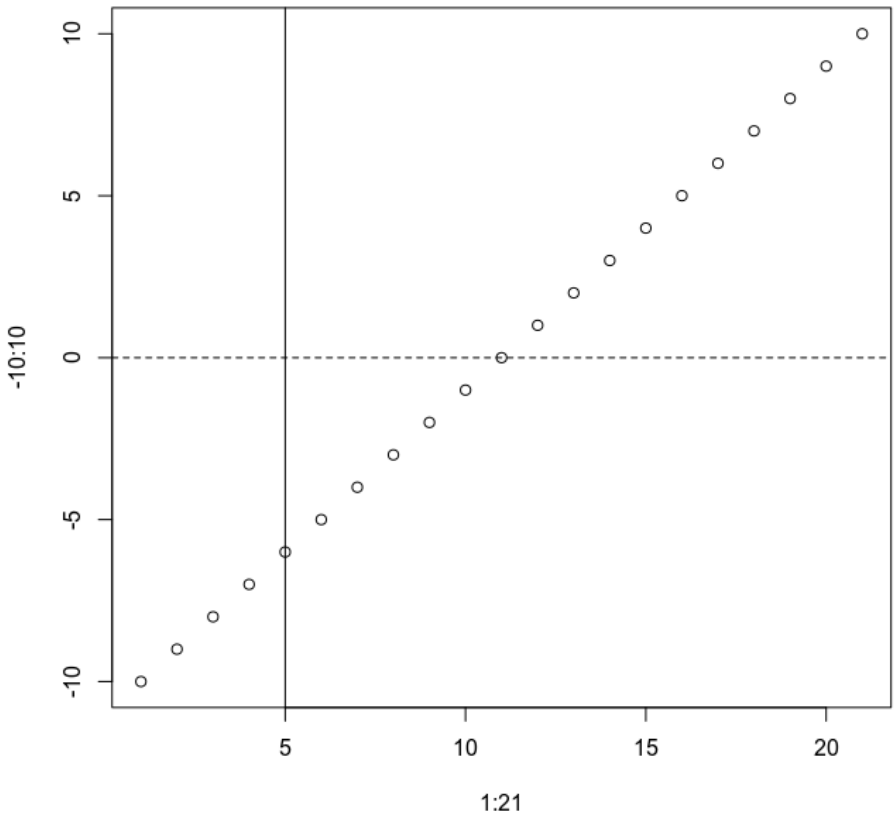


Illustration 35: Plot - abline

13.1.13 Identity

Identify and label a point on a scatter-plot. Use the cursor over a point and it will identify it.

```
> x=rnorm(100,0,2)

> y=rnorm(100,3,3)

> plot(x,y)

> identify(x, y, plot=TRUE)
```

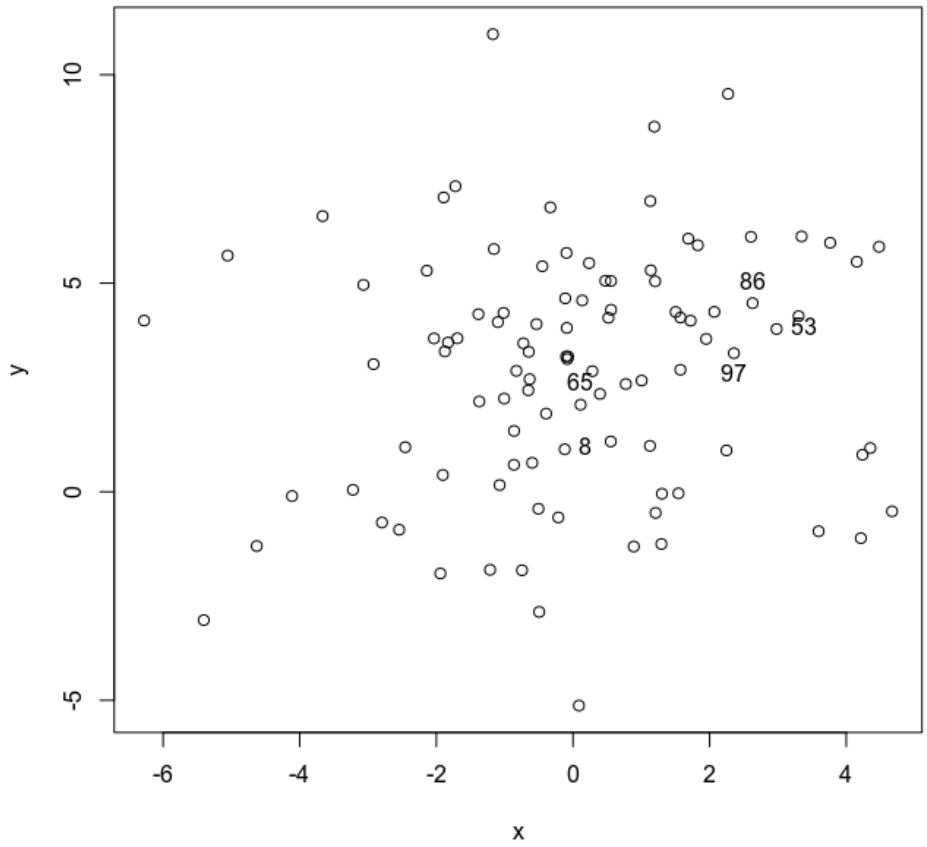


Illustration 36: Plot - identity

13.1.14 Adding equations / Greek letters to graphs

expression(): Creates or tests for objects of mode *expression*.

examples:

```
> plot(c(0,10),c(0,10), type='n')
> text(x=5, y=8, expression(lambda == 1.3))
> text(x=5, y=6, expression(bar(X)[female]==0.55))
> text(x=5, y=4, expression(y[3] ~ x^2 ~ m^-2))
```

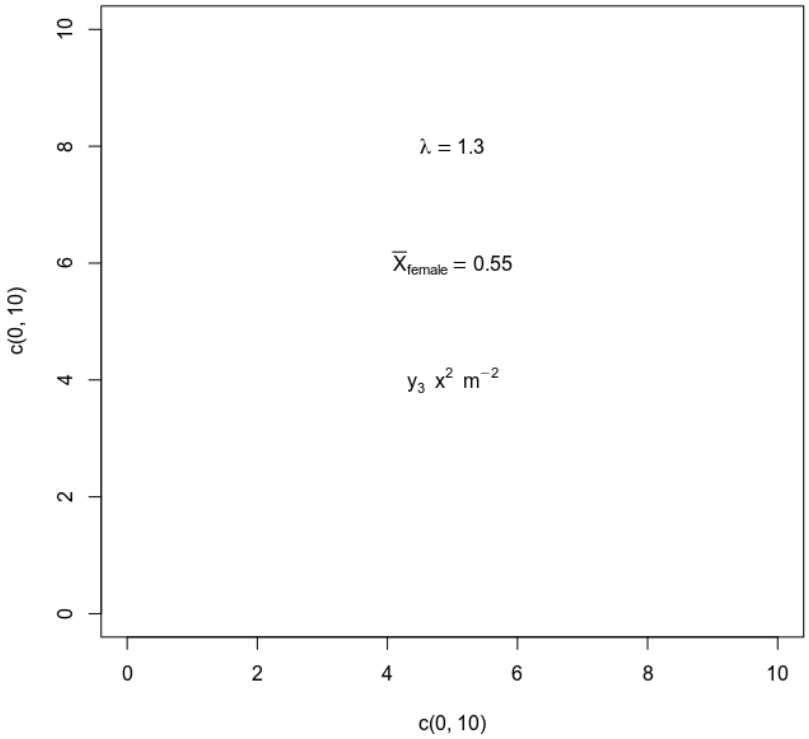


Illustration 37: Plot - expression

13.2 Boxplots

A *box plot* or *boxplot* is a method of representing statistical data on a plot. It consists of a rectangle drawn to represent the second and third quartiles, usually with a vertical line inside to indicate the median value. The lower and upper quartiles are shown as horizontal lines either side of the rectangle.

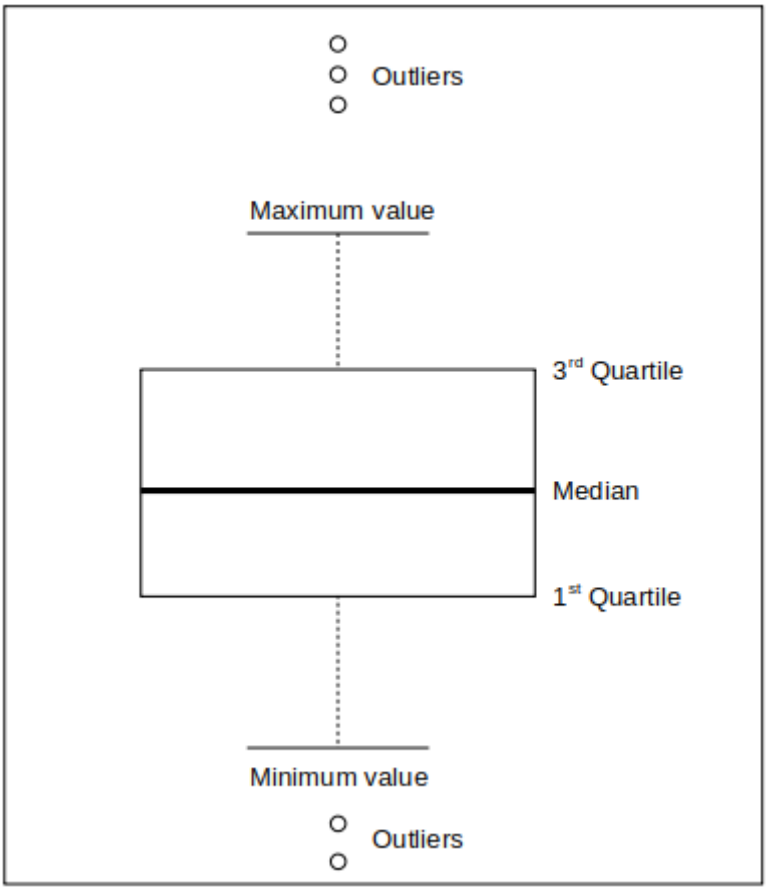


Illustration 38: Boxplot

Figure 37: Plot - boxplot

Considering the following example. As can be seen from the *summary(v)* the minimum and maximum points are marked at *5* and *425*. The median of *152* is flanked by the lower quartile at *69.5* and the upper quartile at *272.5*.

```
> v = c(101,111,112,123,141,152,193,141,
        51,19,43,74,45,26,83,42,65,32,5,
        322,354,385,377,381,314,425,416,
        214,233,234,226,237,248,269,276
       )
```

```
> summary(v)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    5.0    69.5   152.0   184.9   272.5   425.0
```

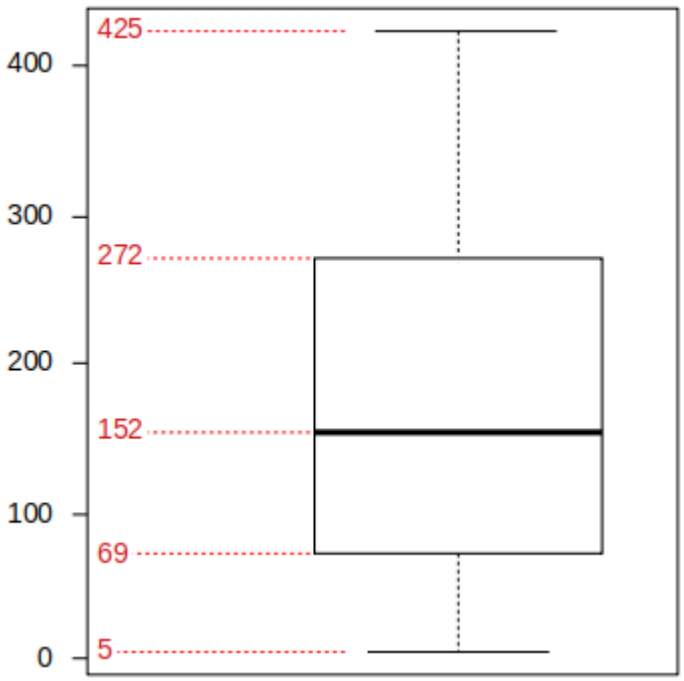


Illustration 39: Boxplot 2

```
> boxplot(v)
```

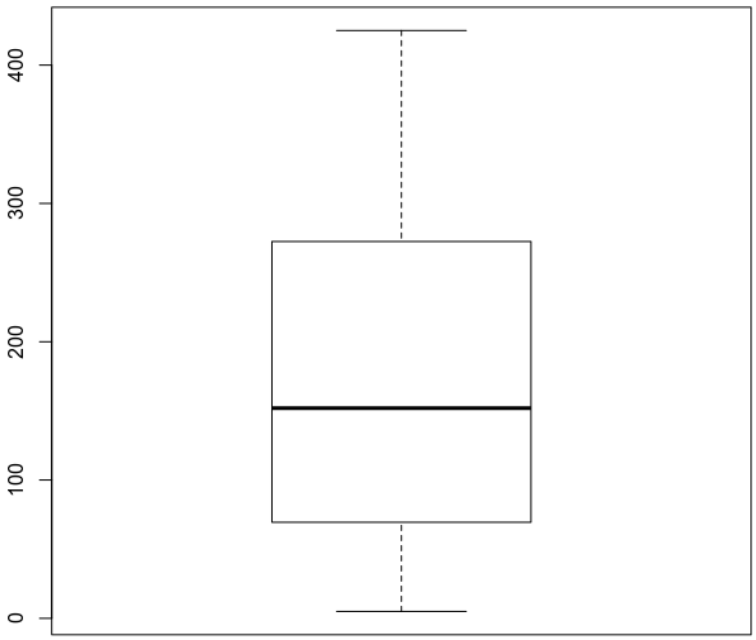


Illustration 40: Boxplot 3

13.3 Saving

13.3.1 Data

Data can be saved in a number of formats, in this case it is saved in *.csv* formatted file. In the example below the plot is saved to: *28-Sep-2018_00.38-graph_name.csv*.

```
> logdir = '/var/log/R/'     # Make sure directory has +w rights
> filename = 'graph_name'    # Make sure directory has +w rights

# Export output
> writeto = paste0(logdir,"/",
                   format(Sys.time(), "%d-%b-%Y_%H.%M-"),
                   filename, ".csv"
                  )

> write.csv(df.normal, writeto)
```

13.3.2 Plots

Plots can also be saved from graphical format to file. In the example below the plot is saved as a *.png* to: *28-Sep-2018_00.35-graph_name.png*. The graphic can also be saved in formats like *.pdf*.

```
logdir = '/var/log/R/'     # Make sure directory has +w rights
filename = 'graph_name'    # Make sure directory has +w rights

# Export output
> writeto = paste0(logdir,"/",
                   format(Sys.time(), "%d-%b-%Y_%H.%M-"),
                   filename, ".png"
                  )

> dev.copy(png, writeto)    # copies the graphics contents
> dev.off()                 #  Shutdown graphic to push to file.
```

13.4 Exercise 1: Making a mess

```
> plot(x=1:100,y=rnorm(100))
```

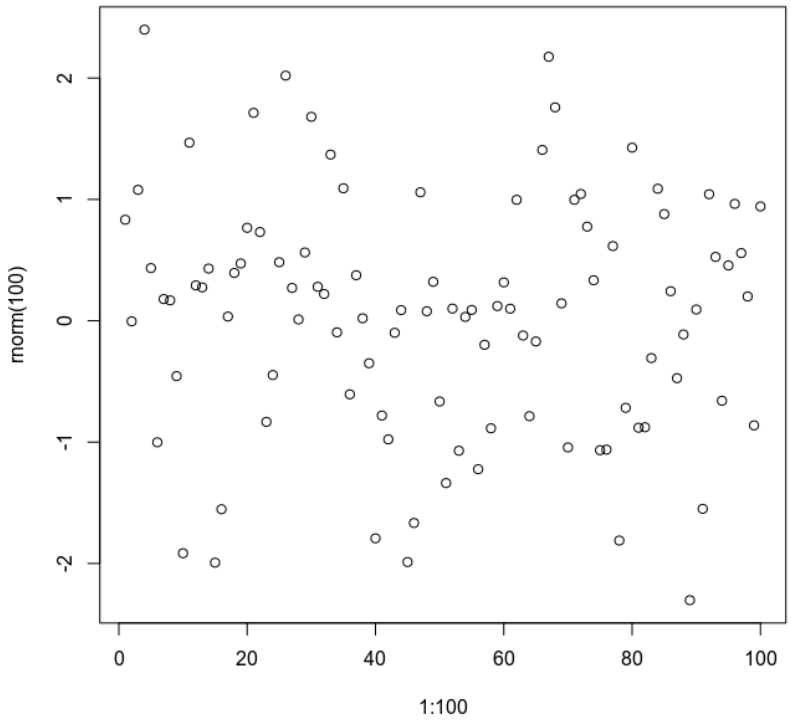


Illustration 41: Plot exercise - start

Taking Illustration 41 apply graphical features liberally to make it as ugly as possible ☺.

Adjust the default background colour.

```
> par(bg = "green")    # Set background to green

> plot(x=1:100,y=rnorm(100),
       pch = 13,              # Change icons to triangles
       cex = 2,               # Enlarge icon to 200%
       lwd = 3,               # Change line width
       col="orange",          # Icon colour
       bg="yellow",           # Icon background colour
       col.main = "orange",   # Change title colour
       col.axis = "red",      # Change axis colour
       col.lab = "white",     # Change label colour
       font.axis = 3,         # Change axis font
       font.lab = 4,          # Change label font
       family = "sans",       # Change font
      )
```

Locate a point on the plot with the *locator()* function. Run the function and click on the plot, it will return the location of the cursor. For the text line the shell will hold until the point is clicked.

```
> locator(1)
$x
[1] -16.85286

$y
[1] 3.984065
```

```
> text(locator(1), "A bit of text is here")

> lines(locator(2),col="white",lty=1,lwd=3)  # Draw a line
> lines(locator(2),col="black",lty=1,lwd=3)  # Draw a line

> locator(2)
$x
[1] 52.10779 51.52666

$y
[1]  2.06091 -2.54633

> points(x=rep(1,25),y=1:25, type= 'n')

# Draw a line
> segments(52.10779, 2.06091, 51.52666, -2.54633,col="red",lty=2,lwd=4)

# Draw a line
> lines(c(4.455207, 96.273600), c(-0.33893, -0.30497) ,col="yellow",lty=2,lwd=4)

> title("Is it Saint Patricks Day")  # Add a title

> polygon(c(1,1:7,7),                # Add a polygon
          c(0,1,2,1,2,1,2,1,0),
          col='blue', lwd=3
         )
```

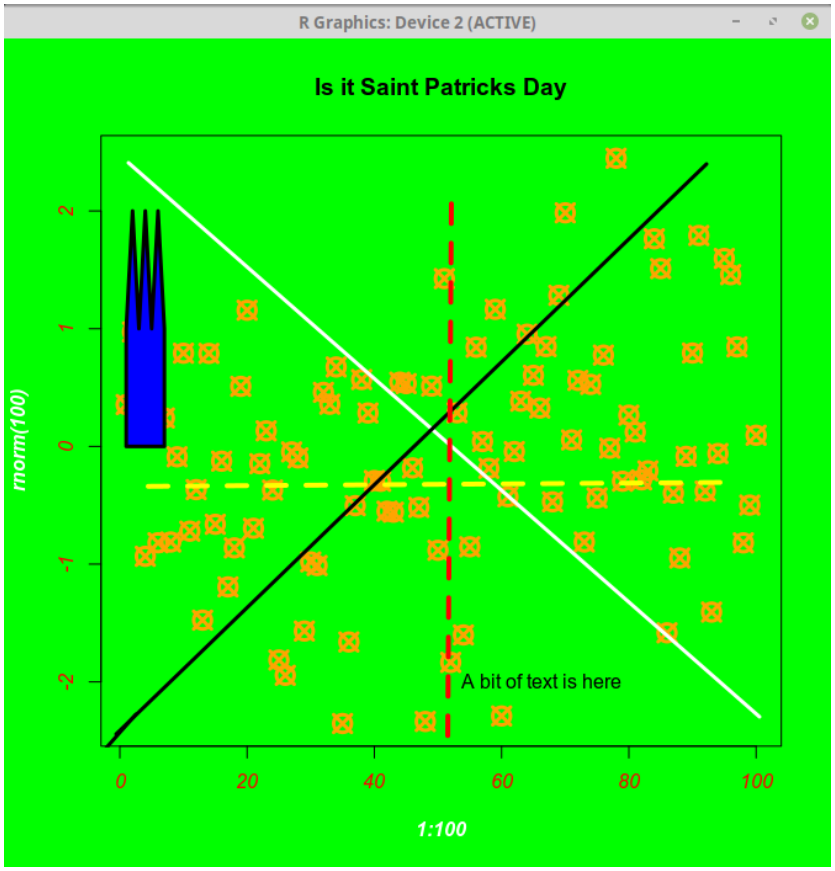


Illustration 42: Plot - mess

13.5 Exercise 2: Create a boxplot

As has been demonstrated in section 13.2 Boxplots *R* comes with a *boxplot()* function as in Illustration 43. Assuming there is no such function create a boxplot for the output of the function *rnorm(100,5,1.5)* using the other tools available.

```
> a = rnorm(100,5,1.5)
> boxplot(a)
```

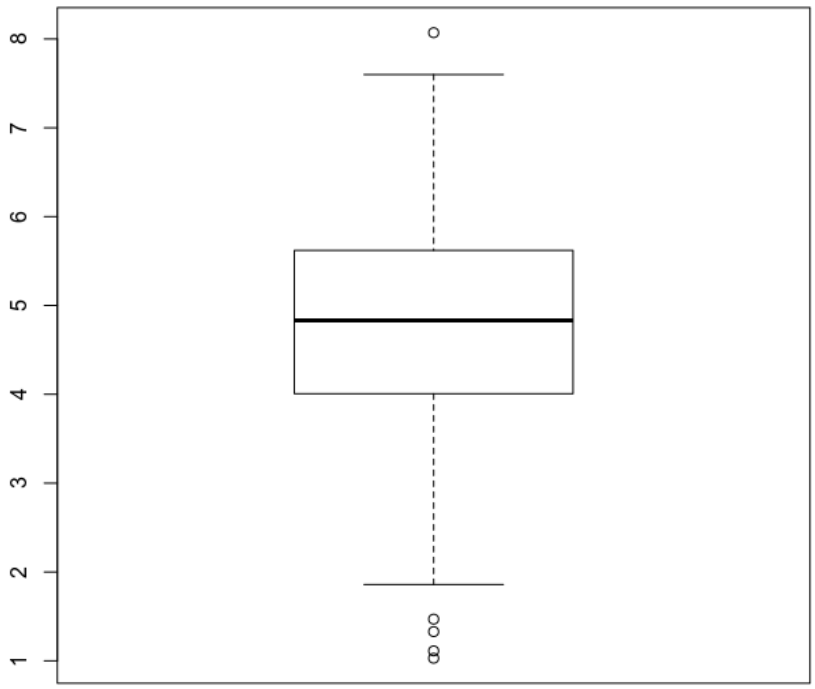


Illustration 43: Boxplot exercise

Answer:

```
> a = rnorm(100,5,1.5)
> mean_a = mean(a)
> sd_a = sd(a)
> min_a = min(a)
> max_a = max(a)

> plot(c(0, 0, min_a, min_a),     # Plot area
       c(0,max_a,max_a,0),
       ylim = c(0,10), xaxt = 'n', yaxt = 'n',
       type = 'n', xlab = '', ylab = ''
      )

> segments(1, min_a,      # Dotted line line
         1, max_a,
         col="black",lty=2,lwd=1
       )

> segments(0.9,max_a,        # Top line
         1.1, max_a,
         col="black",lwd=1
         )

> segments(0.9,min_a,    # Bottom line
         1.1,min_a,
         col="black",lwd=1
        )

> polygon(c(0.8, 1.2, 1.2, 0.8),   # Draw polygon
          c(mean_a - sd_a,mean_a - sd_a, mean_a + sd_a,mean_a + sd_a),
          border = "black",
          col = "white",
          lwd=1
         )

> segments(0.8,mean_a, 1.2, mean_a,     # Draw middle line
         col="black",lwd=4
        )

> axis(2, at=c(1,2,3,4,5,6,7,8), labels=c(1,2,3,4,5,6,7,8))
```

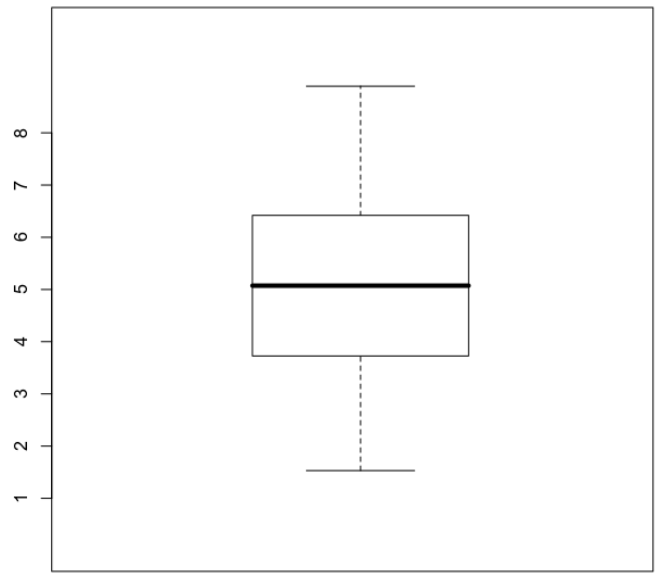


Illustration 44: Boxplot answer

13.6 Multiple graphs

13.6.1 par() function again

With the *par()* function, the options *mfrow=c(nrows, ncols)* can be included to create a matrix of *nrows x ncols* plots that are filled in by row. *mfcol=c(nrows, ncols)* fills in the matrix by columns. To plot the model *model.1* in the example will shuffle through four plots.

- Residuals vs Fitted
- Normal Q-Q
- Scale-Location
- Residuals vs Leverage

```
> x = c(1,2,2,3,2,3,4,3,4,5,3)

> y = c(4,8,6,3,5,7,9,2,1,7,4)

> model.1 = lm(y ~ x)

> plot(model.1)

Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
```

However by adjusting the graphical parameters it is possible to create a top-level graphical plot area that will include the four plots on the one main plot area. As there are four plots then a 2 x 2 main plot frame is required.

```
> x = c(1,2,2,3,2,3,4,3,4,5,3)

> y = c(4,8,6,3,5,7,9,2,1,7,4)

> model.1 = lm(y ~ x)

> par(mfrow=c(2,2))   # Adding this line creates the top-level plot area

> plot(model.1)
```

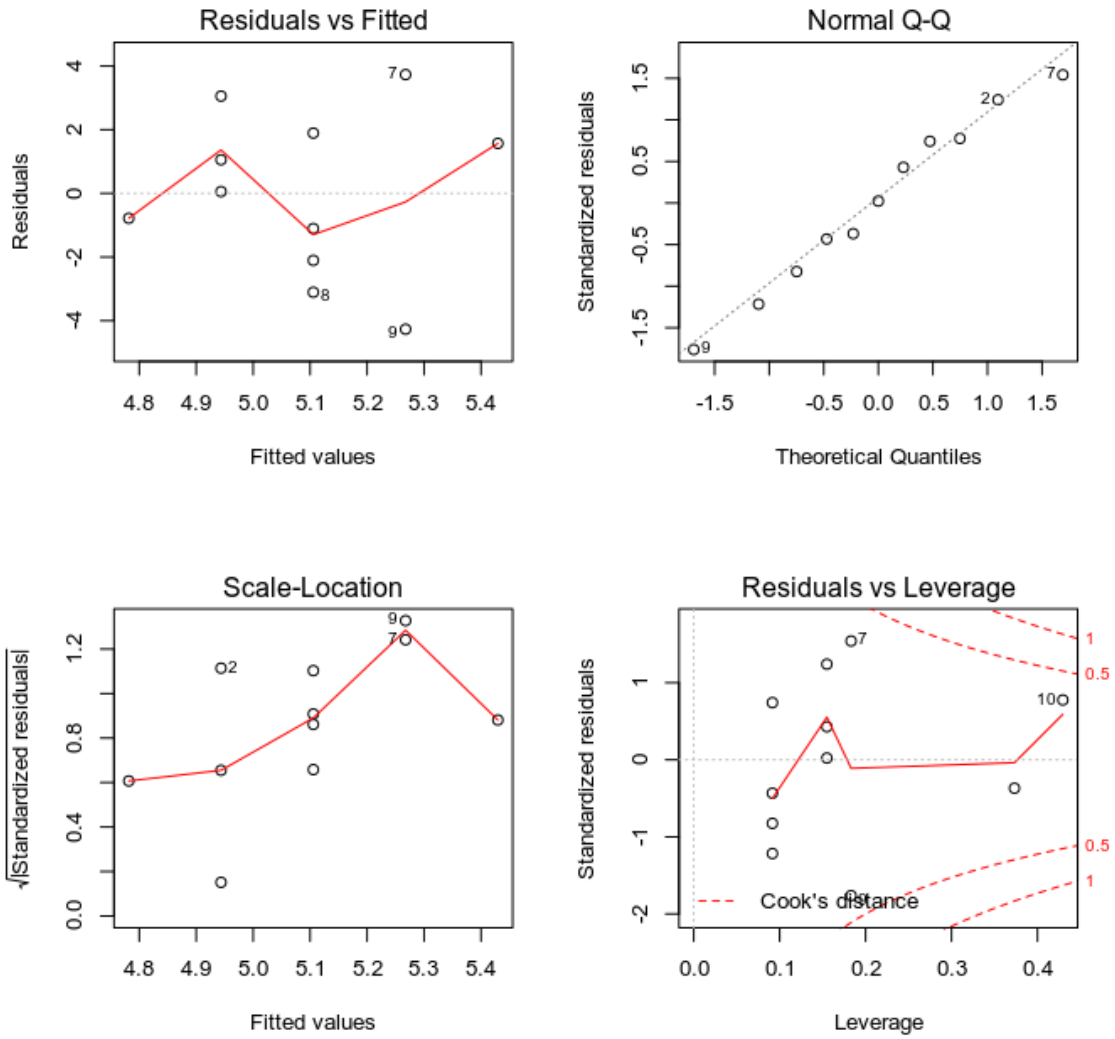


Illustration 45: Multiple graphs

13.7 Exercise 3a: Setting graph parameters

Using these data, create a publication quality graph like this.

HINT: to get the axis tick marks on the inside use the *tck* argument in your plot call (e.g. *tck=0.03*).

```
> x = 1:20

> y = seq(from=3, to=7, length.out=20)+rnorm(20,0,2)
```

Answer:

```
> x = 1:20
> y = seq(from=3, to=7, length.out=20)+rnorm(20,0,2)
> plot(x,y, type='n',
      xlab = 'Years since establishment',
      ylab = 'Population size (x 1000)',
      xaxt = 'n', yaxt = 'n',
      bty = 'l',
      xlim = c(0,20),
      ylim = c(0,10),
      font.lab=2
     )

> points(x,y, pch=21, bg="black", cex=2)

> axis(1, seq(0,20,5),las=2, tck = 0.02)

> axis(2, seq(0,10,2), tck = 0.02)
```

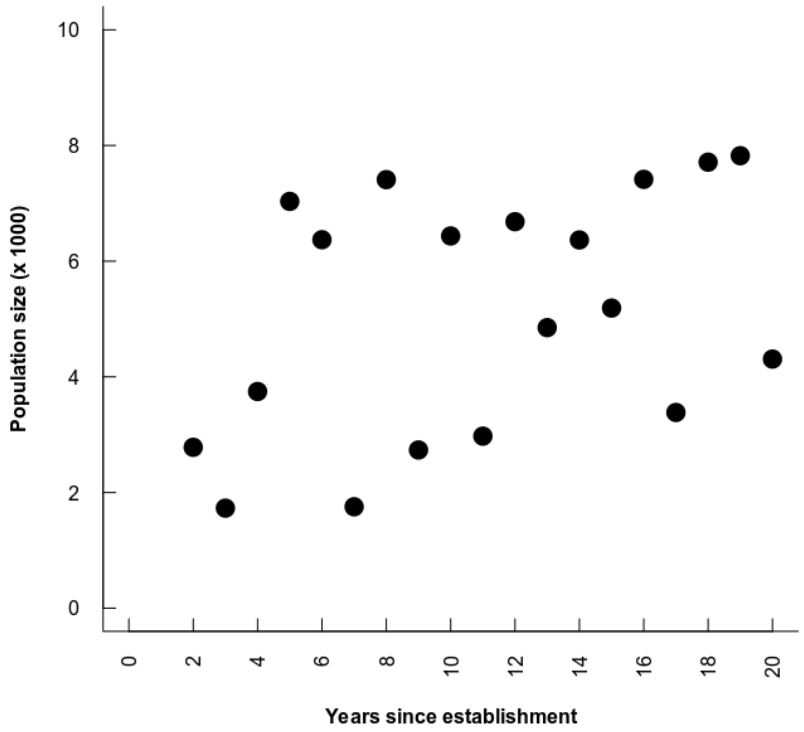


Illustration 46: Exercise - setting graph parameters

13.8 Exercise 3b: Setting graph parameters

Add a second plot using these additional data.

```
> x = 1:20
> proportion.females = runif(20,0.3,0.7)
> y = proportion.females
> plot(x,y, type='l',
       xlab = 'Years since establishment',
       ylab = 'Proportion of females in the population',
       xaxt = 'n', yaxt = 'n',
       bty = 'l',
       lty = 2,
       xlim = c(0,20),
       ylim = c(0,1),
       font.lab=2
      )

> axis(1, seq(0,20,5),las=2, tck = 0.02)

> axis(2, seq(0,1,0.2),tck = 0.02)

> text(x=1, y=0.98, "b", cex=1.5)

> text(x=10, y=0.15, expression(bar(X)[female]==0.55), cex=1.3)
```

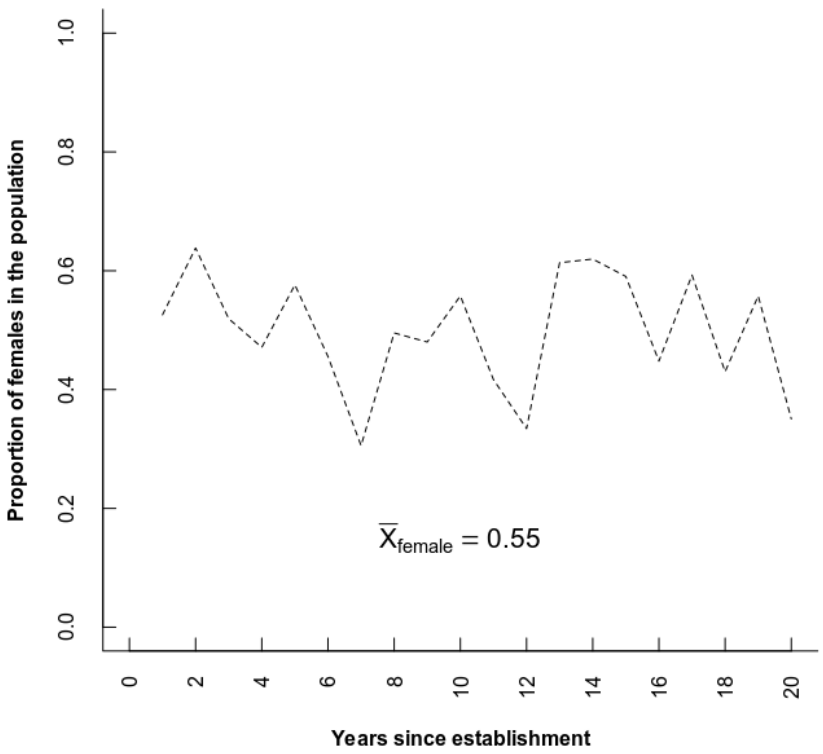


Illustration 47: Exercise - setting graph parameters 2

Create a multiple frame to hold both graphs.

mtext(): Write Text into the Margins of a Plot.

```
# Data to be plotted
> x = 1:20
> y = seq(from=3, to=7, length.out=20)+rnorm(20,0,2)
> z = runif(20,0.3,0.7)

# First graphic area (left side)
> par(fig=c(0,0.5,0,1))

# Plot the points on first graph
> plot(x,y, type='n',
       xlab = '',
       ylab = 'Population size (x 1000)',
       xaxt = 'n', yaxt = 'n',
       bty = 'l',
       xlim = c(0,20),
       ylim = c(0,10),
       font.lab=2
      )

# Redefine the points
> points(x,y, pch=21, bg="black", cex=2)

# Add the first graph axis
> axis(1, seq(0,20,5),tck = 0.02)
> axis(2, seq(0,10,2), tck = 0.02)

# Add the "a" and the function term to 1st graph
> text(x=1, y=9.8, "a", cex=1.5)
> text(x=10, y=9, expression(lambda == 1.3), cex=1.5)

# Second graphic area (right side)
> par(fig=c(0.5,1,0,1), new=T)

# Plot the dashed line
> plot(x,z, type='l',
       xlab = '',
       ylab = 'Proportion of females in the population',
       xaxt = 'n', yaxt = 'n',
       bty = 'l',
       lty = 2,
       xlim = c(0,20),
       ylim = c(0,1),
       font.lab=2
      )

# Add in the axis graphs
> axis(1, seq(0,20,5),tck = 0.02)
> axis(2, seq(0,1,0.2),tck = 0.02)

# Add the "b" and the function term to 2nd graph
> text(x=1, y=0.98, "b", cex=1.5)
> text(x=10, y=0.15, expression(bar(X)[female]==0.55), cex=1.3)

# Add the common text under both graphs
> mtext("Years since establishment", side = 1, line = -2, outer = TRUE, cex=1.3)
```

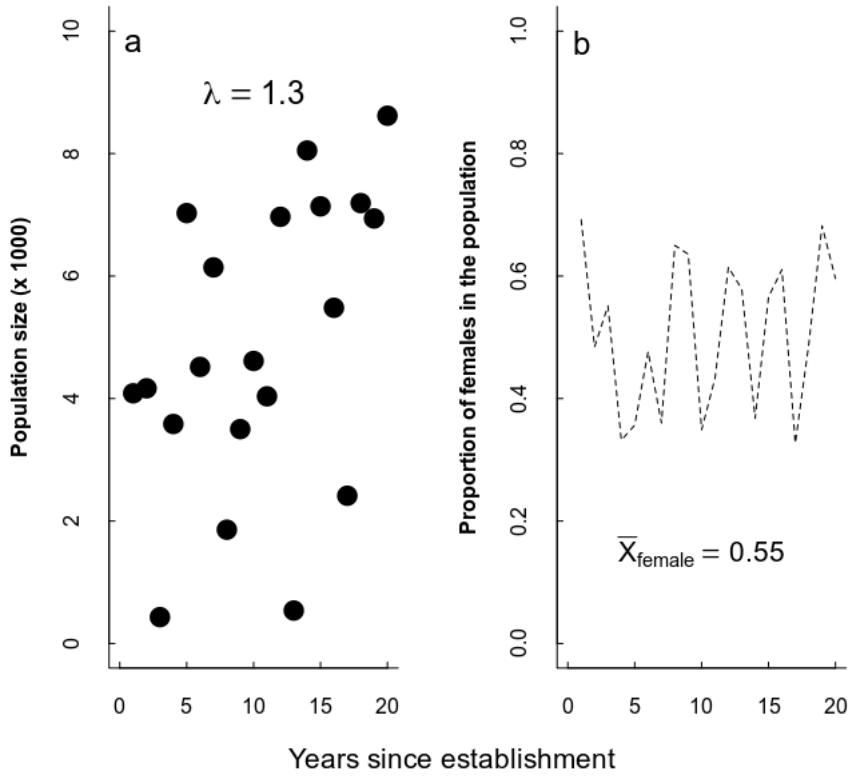


Illustration 48: Exercise - setting graph parameters 3

13.9 Exercise 4a: Prediction plots

Using the model *lm(begging~sex+food+sex*food, data = owl)*

Extract predictions (with their standard errors) for both males and females, for a range of food values from 20-30 using the *predict()* function.

```
# Add libraries
> library(scales)

# Import the data
> owl = read.csv('owl_data.csv')

> owl.lm = lm(begging ~ sex + food + sex * food, data = owl)

> male_predict = data.frame(sex=rep('Male',11), food=10:20)
> female_predict = data.frame(sex=rep('Female',11), food=10:20)

> male_out = predict.lm(owl.lm, male_predict, se.fit=T)
> female_out = predict.lm(owl.lm,female_predict, se.fit=T)

# Plot 1

# Generate the first plot
> plot(x=seq(10,20),y=male_out$fit,type='n',
       xlab='Time since last meal',
       ylab='Begging rate',
       bty='l', ylim=c(8,25), cex.lab=1.2
      )

# Add lines to plot
> lines(x=seq(10,20),male_out$fit,
        type='b', pch=16, col='blue'
       )

> lines(x=seq(10,20),female_out$fit,
        type='b', pch=16, col='red'
       )

# Plot 2: Add Standard Errors (SE) lines on the plot

# Add SE lines for males
> lines(x=seq(10,20),male_out$fit+male_out$se.fit,
        type='l', col='blue', lty=2
       )

> lines(x=seq(10,20),male_out$fit-male_out$se.fit,
        type='l', col='blue', lty=2
       )

# Add SE lines for females
> lines(x=seq(10,20),female_out$fit+female_out$se.fit,
        type='l', col='red', lty=2
       )

> lines(x=seq(10,20),female_out$fit-female_out$se.fit,
        type='l', col='red', lty=2
       )

# Plot 3: now add SE polygons

# Add SEs for males
> polygon(x=c(10:20, 20:10),
          y=c(male_out$fit+male_out$se.fit,
          rev(male_out$fit-male_out$se.fit)),
          col=alpha("blue", 0.3), border="blue"
```

```
        )

# Add SEs for females
> polygon(x=c(10:20, 20:10),
         y=c(female_out$fit+female_out$se.fit,
         rev(female_out$fit-female_out$se.fit)),
         col=alpha("red", 0.3), border="red"
        )

# Add text to the plot
> text(16,19,'Male', cex=1.5, col='blue')
> text(15,9,'Female', cex=1.5, col='red')
```

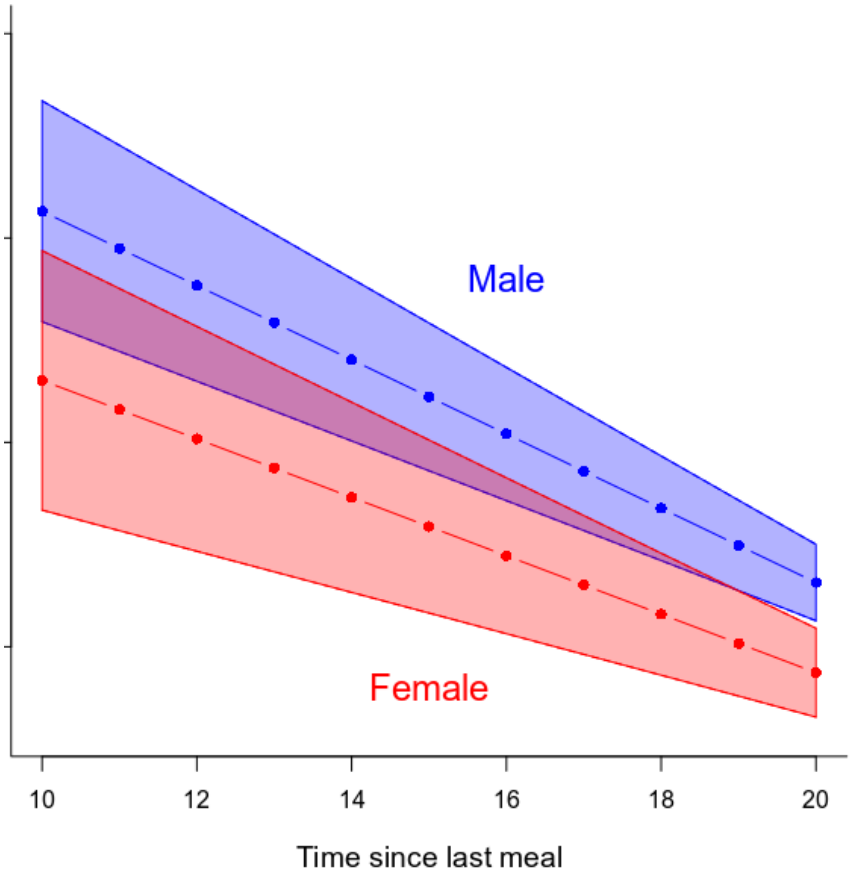


Illustration 49: Exercise - prediction plots

13.10 Exercise 4b: Prediction plots

Modify the previous model to a Poisson distribution.

*glm(begging~sex+food+sex*food, data=owl, family=poisson)*

Repeat exercise 4a, but using the predictions from the model based on a poisson distribution.

HINT:check if your predictions are at the log scale and need to be exponentiated. check the predict function argument *type*. This needs to be set to *response*.

```
# Add libraries
> library(scales)

# Import the data
> owl = read.csv('owl_data.csv')

> owl.lm = glm(begging ~ sex+food+sex*food, data=owl, family=poisson)

> male_predict = data.frame(sex=rep('Male',11), food=10:20)
> female_predict = data.frame(sex=rep('Female',11), food=10:20)

> female_out = predict.glm(owl.lm, male_predict, se.fit=T, type="response")
> male_out = predict.glm(owl.lm,female_predict, se.fit=T, type="response")

# Plot 1

# Generate the first plot
> plot(x=seq(10,20),y=male_out$fit,type='n',
       xlab='Time since last meal',
       ylab='Begging rate',
       bty='l', ylim=c(0,60), cex.lab=1.2
      )

# Add lines to plot
> lines(x=seq(10,20),male_out$fit,
        type='b', pch=16, col='blue'
       )

> lines(x=seq(10,20),female_out$fit,
        type='b', pch=16, col='red'
       )

# Plot 2: Add Standard Errors (SE) lines on the plot

# Add SE lines for males
> lines(x=seq(10,20),male_out$fit+male_out$se.fit,
        type='l', col='blue', lty=2
       )

> lines(x=seq(10,20),male_out$fit-male_out$se.fit,
        type='l', col='blue', lty=2
       )

# Add SE lines for females
> lines(x=seq(10,20),female_out$fit+female_out$se.fit,
        type='l', col='red', lty=2
       )

> lines(x=seq(10,20),female_out$fit-female_out$se.fit,
        type='l', col='red', lty=2
       )
```

```
# Plot 3: now add SE polygons

# Add SEs for males
> polygon(x=c(10:20, 20:10),
          y=c(male_out$fit+male_out$se.fit,
          rev(male_out$fit-male_out$se.fit)),
          col=alpha("blue", 0.3), border="blue"
         )

# Add SEs for females
> polygon(x=c(10:20, 20:10),
          y=c(female_out$fit+female_out$se.fit,
          rev(female_out$fit-female_out$se.fit)),
          col=alpha("red", 0.3), border="red"
         )

# Add text to the plot
> text(16,40,'Female', cex=1.5, col='red')
> text(15,9,'Male', cex=1.5, col='blue')
```

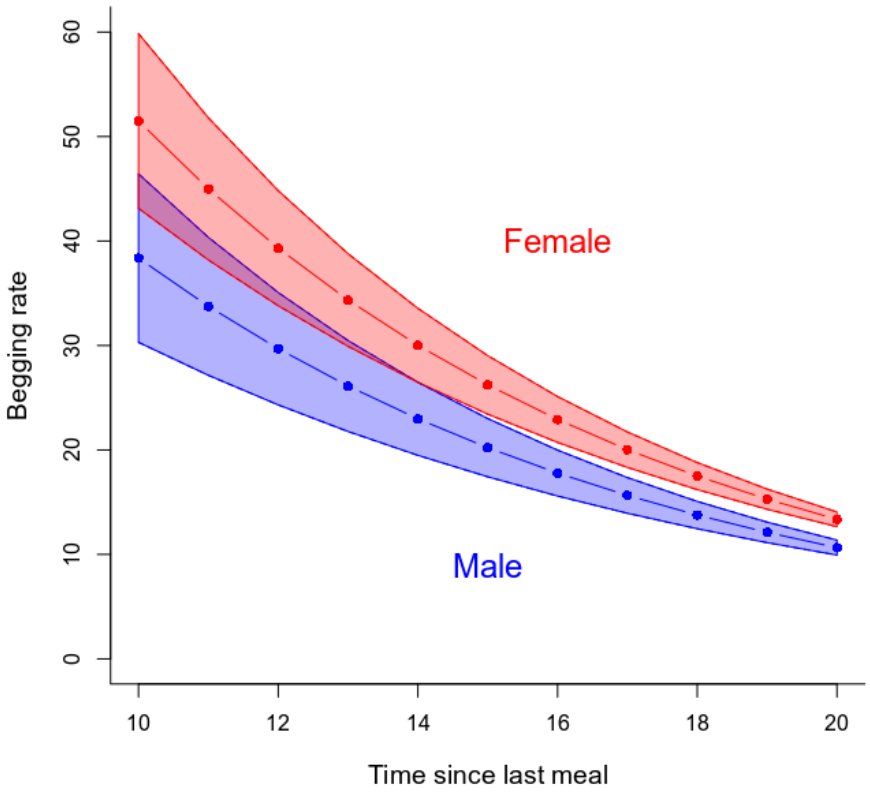


Illustration 50: Exercise - prediction plots 2

13.11 Exercise 5: Visualising plot data

Import the *bird_egg.csv* file.

Plot the relationship between eggs (y) and age (x) for clutch 1 (blue points) & clutch 2 (red points) and save it as a pdf file.

HINT: *jitter()* could help visualise the points here eggs is the response variable (y) and age the explanatory (x).

EXTRA: extract predictions from a model where age and clutch (and their interaction) explain the number of eggs. Plot these predictions with their standard errors over the scatter-plot above.

```
> install.packages('scales')

# Add libraries
> library(scales)

# Import the data
> bird = read.csv('bird_egg.csv')

> names(bird)
#[1] 'individual'   'year'   'clutch'   'age'    'eggs'   'dist_food'
#[7] 'fail_fledge'

# Plot the points and
> x1 = bird[bird[,3]==1,4]  # Plot the 'age' for the 1st clutch
> y1 = bird[bird[,3]==1,5]  # Plot the 'eggs' for the 1st clutch
> x2 = bird[bird[,3]==2,4]  # Plot the 'age' for the 2nd clutch
> y2 = bird[bird[,3]==2,5]  # Plot the 'eggs' for the 2nd clutch

# Plot the eggs as a DV for age as the IV for 1st clutch
> plot(x1,y1,col='blue',
         xlab='Age', ylab='Eggs'
       )

# Plot the eggs as a DV for age as the IV for 2nd clutch
> points(x2,y2, col='red')

# Add a small amount of noise to the vector.
> points(jitter(x1),jitter(y1), col='blue')
> points(jitter(x2),jitter(y2), col='red')

# Get data for clutch 1 & 2 (poisson)
> bird1 = bird[bird[,3] < 3,]
> mod1 = glm(eggs~age * as.factor(clutch),
             data=bird1, family='poisson'
            )
> summary(mod1)

# Extract dataframe for 'age' and 'clutch'
> new_data1 = data.frame(age=1:10, clutch=rep(1,10))
> new_data2 = data.frame(age=1:10, clutch=rep(2,10))

# Get predictions
> pred_clutch1 = predict.glm(mod1, new_data1, se.fit=T, type='response')
> pred_clutch2 = predict.glm(mod1, new_data2, se.fit=T, type='response')

# Add clutch1 prediction

# Add lines
> lines(x = 1:10, y = pred_clutch1$fit,
        lty = 1, lwd = 2, col = 'blue'
```

```
       )

# Add blue filled in pologon area
> polygon(x = c(1:10, 10:1),
          y = c(pred_clutch1$fit+pred_clutch1$se.fit,
          rev(pred_clutch1$fit-pred_clutch1$se.fit)),
          col = alpha('blue',0.3), border='blue'
         )

# Add clutch2 prediction

# Add lines
> lines(x = 1:10, y = pred_clutch2$fit,
        lty = 1, lwd = 2, col = 'red'
        )

# Add red filled in pologon area
> polygon(x = c(1:10, 10:1),
          y = c(pred_clutch2$fit+pred_clutch2$se.fit,
          rev(pred_clutch2$fit-pred_clutch2$se.fit)),
          col = alpha('red',0.3), border='red'
         )
}
# Add labels
> text(x = 6, y = 5.5, 'Clutch 1',
       cex = 1.5, col = 'blue'
      )

> text(x = 6, y = 2, 'Clutch 2',
       cex = 1.5, col = 'red'
      )
```

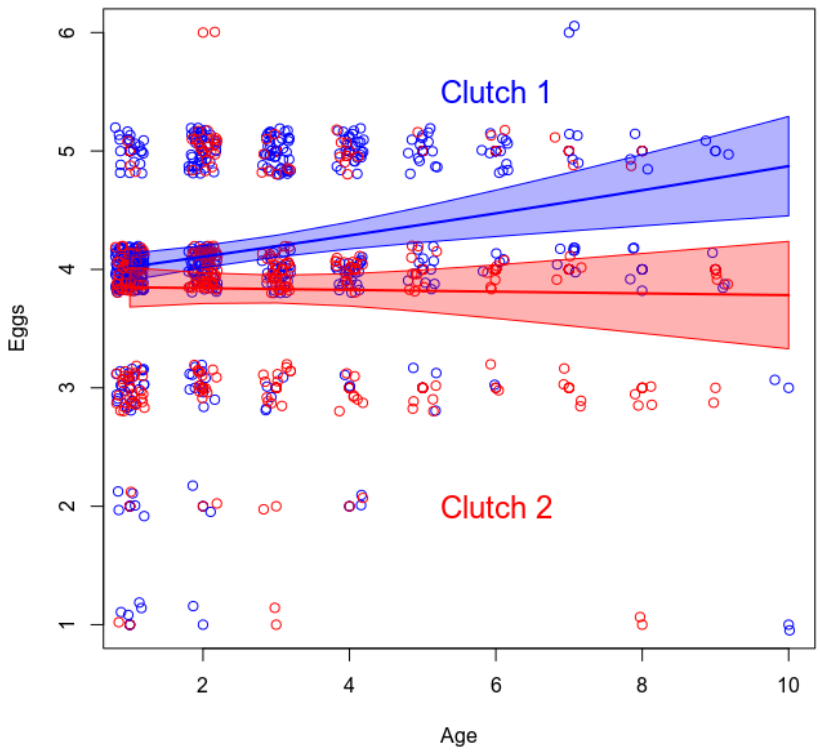


Illustration 51: Exercise - visualising plot data

13.12 Exercise 6: Pretty plot

From the data *xy_data_quadplotex.csv* produce a graph. The model is *y ~ x + x 2*.

HINT: Get the fitted values for the model prediction.

```
# Look at the data
> read.csv('xy_data_quadplotex.csv')
    x          y
1   1  3.9257408
2   2  1.9763139
3   3  3.1513985
4   4  2.6076583
5   5  5.0019918
6   6  5.7526961
7   7  6.3944896
8   8  5.4130619
9   9  9.4346628
10 10  9.7635863
11 11  8.9396002
12 12  7.2992594
13 13  9.1645094
14 14  8.1535297
15 15  7.9302138
16 16  9.4534872
17 17  6.2583997
18 18  6.5865369
19 19  5.7238197
20 20  6.0263360
21 21  3.0479905
22 22  3.3684445
23 23  4.5477625
24 24  3.1443438
25 25 -0.3731705
26 26  0.9693107
27 27 -5.7287515
28 28 -3.4355719
29 29 -2.9714692
30 30 -7.3180748

# Import the data
> xy = read.csv('xy_data_quadplotex.csv')

# Create the initial plot with values
> plot(x,y)

# Create linear model
> mod.1 = lm(xy$y ~ xy$x + I(xy$x^2))
> y.fit = fitted(mod.1)

# Draw the fit line on plot
> lines(xy$x,y.fit, lwd=2)


> get.y = matrix(c(xy$x,y.fit), ncol=2)

# Show area under curve for range 1 - 7
> x1.range <- 1:7
> y1.range <- get.y[get.y[,1]>=1 & get.y[,1]<=7,2]

# Show area under curve for range 7 - 20
> x2.range <- 7:20
> y2.range <- get.y[get.y[,1]>=7 & get.y[,1]<=20,2]

# Show area under curve for range 20 - 30
> x3.range <- 20:30
```

```
> y3.range <- get.y[get.y[,1]>=20 & get.y[,1]<=30,2]

# Add shading area under the curve for x - 1 - 7
> polygon(c(1,x1.range,7),
          c(min(y),y1.range,min(y)), density=20,
          col="green", angle=45, border=NA
         )

# Add shading area under the curve for x - 7 - 20
> polygon(c(7,x2.range,20),
          c(min(y),y2.range,min(y)), density=20,
          col="red", angle=45, border=NA
         )

# Add shading area under the curve for x - 20 - 30
  > polygon(c(20,x3.range,30), c(min(y),y3.range,min(y)), density=20, col="yellow",
angle=45, border=NA)

# Add labeling to each shaded area
> text(3.5,-3, "Treatment\nA", cex=1)
> text(12.5,-3, "Treatment\nB", cex=1)
> text(24,-3, "Treatment\nC", cex=1)
```

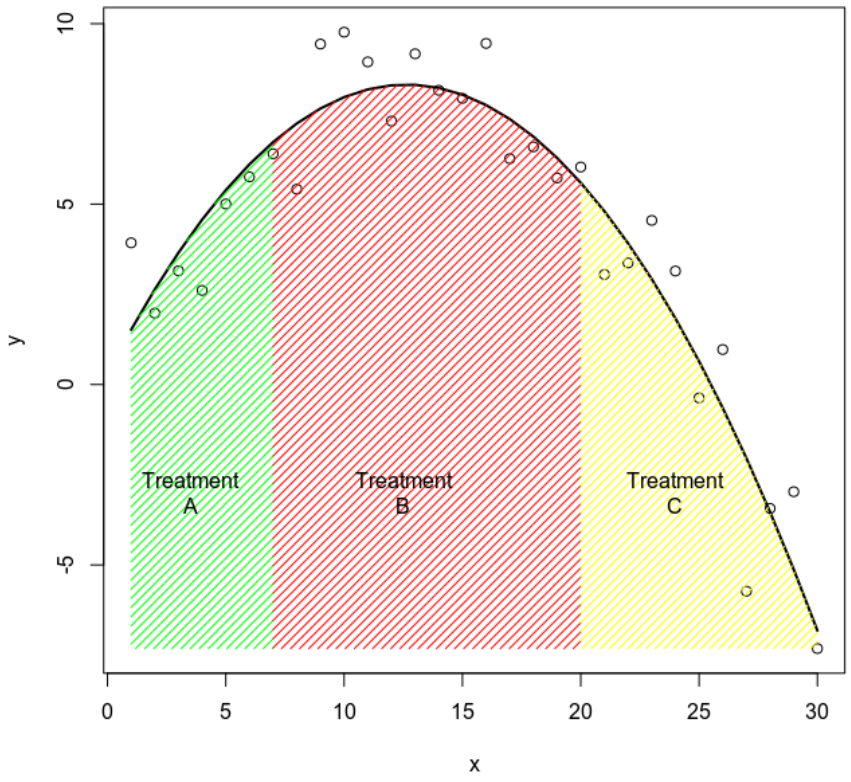


Illustration 52: Exercise - pretty plot

13.13 Exercise 7: Icon and colour table

```
# Add the  plot
> plot(x = c(0,1.2),y = c(0,25),
       col = "white", xlab = "", ylab = "",
       xaxt = "n",yaxt = "n"
      )

# Add points for icons
> points(x = rep(0.1,25),
         y = 1:25,pch = 1:25,
         cex = 1.2
        )

# Add text for icons
> text(x = rep(0.1,25),
       y = 1:25,paste("pch",1:25),
       pos = 4, offset = 1
      )

# Add points for colour
> points(x = rep(0.4,8),
         y = 10:17,col = 1:8,
         pch = 16, cex = 3
        )

# Add text for colour
> text(x = rep(0.4,8),y = 10:17,
       paste("colour",1:8),pos = 4,
       offset = 1
      )

# Add lines
> segments(x0 = rep(0.7,20),y0 = 11:16,
           x1 = rep(0.9,20),lty = 1:6,
           lwd = 1.3,col = "black"
          )

# Add text to lines
> text(x = rep(0.9,20),y = 11:16,
       paste("lty",1:6),pos = 4,
       offset = 1
       )

# Add title text

> title_msg = "Exercise 7: reproduce & save this plot for future reference"

> mtext(title_msg,
        side = 3, outer = TRUE, line = -2.2:-2, font = 2
       )
```

Exercise 7: reproduce & save this plot for future reference

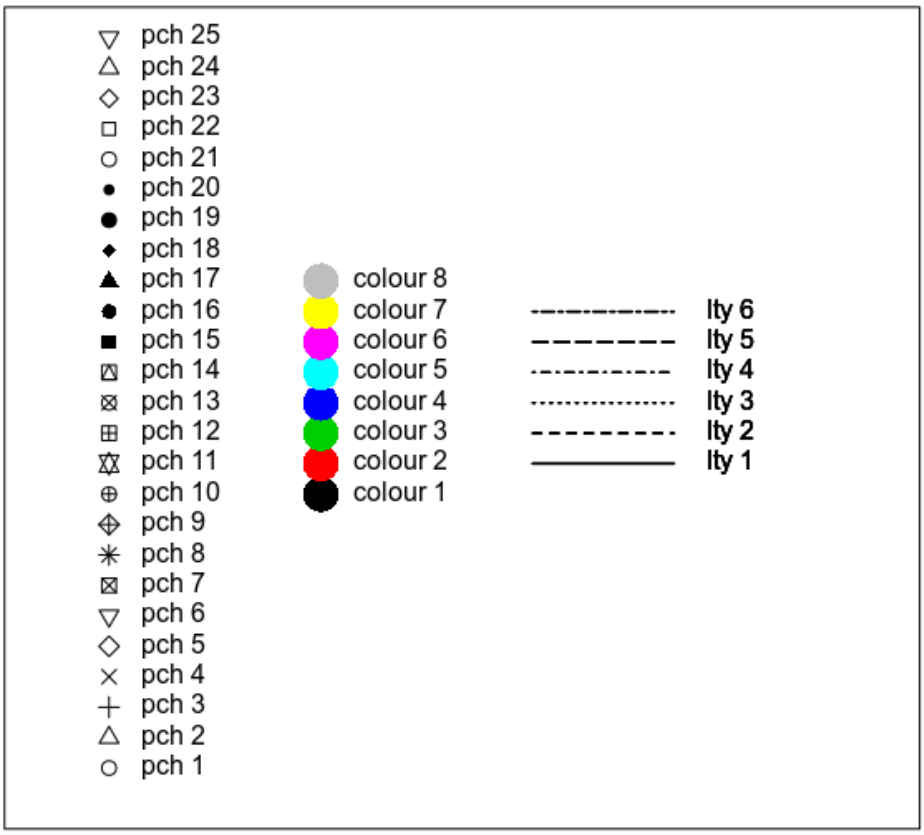


Illustration 53: Exercise - useful table

14. Generalised Linear Mixed Models (GLMM)

Taking the earlier prediction:

```
> owl = read.csv('owl_data.csv')

> owl.lm = lm(begging ~ sex + food + sex * food, data = owl)
```

The assumption was that the residuals were random and had no relationship with each-other. However it could be observed for example that there is a higher possibility of the residuals below the model line being say male in blue and above the line being female. Thus there is non-independence of the residuals and this needs to be accounted for. One way is to do separate ANCOVA for set, male and female.

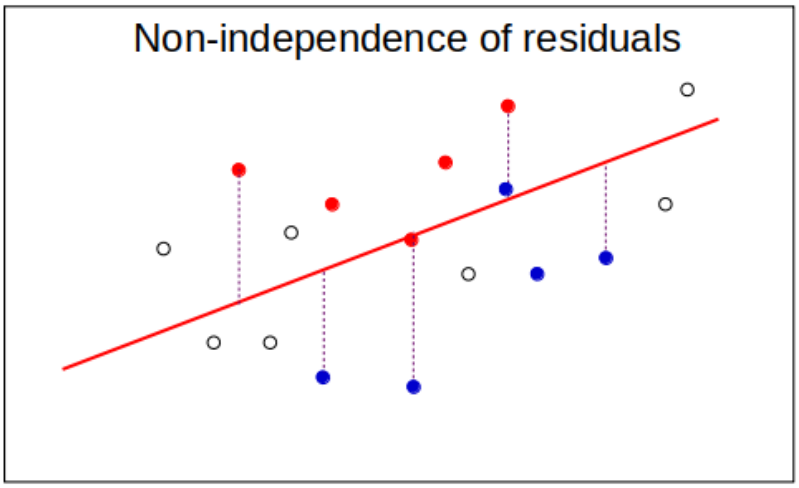


Illustration 54: Non-independence of residuals

However this could get impractical quite quickly. Imagine if there were samples taken from a number of sources and for practical reasons there were more samples taken from certain sources than others. Now there is non-independence between the samples taken from each source and because of the different ratios of samples from each it must be accounted for in the linear models.

- **nlme:** Non-linear Mixed-Effects Models (NLME). This generic function fits a non-linear mixed-effects model in the formulation described in Lindstrom and Bates (1990) but allowing for nested random effects. The within-group errors are allowed to be correlated and/or have unequal variances.
- **nlme:** Fit Linear Mixed-Effects Models (LMM). Fit a LMM to data, via the Restricted (or Residual, or Reduced) Maximum Likelihood (REML).

Import the *owl_data.csv* file. There are four models. The first two are standard Linear Models. The second two models fit the same model as the *lm()* function, however they must have at least one random effect, in this case the *(1|nest)* argument. If you want to fit another type of distribution you use *glmer()* and set the family argument. It is demonstrated below for the family *poisson* however in this case it would fail to converge as the dataset is not suitable for that distribution.

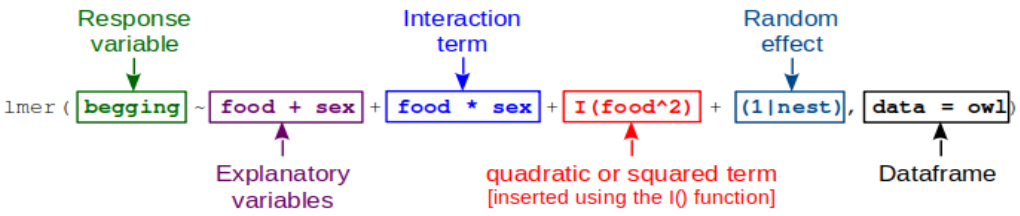


Illustration 55: Generalised Mixed Models

```
# Install packages (Unhash the first time to install packages)
# install.packages('nlme')
# install.packages('lme4')

# Load libraries
> library(nlme)
> library(lme4)

# Generate models
> mod.lm = lm(begging ~ food + sex + food * sex + I(food^2), data = owl)

> mod.glm = glm(owl.lm = lm(begging ~ sex + food + sex * food, data = owl)(food^2),
                data=owls, family=gaussian
               )

> mod.lme = lme(owl.lm = lm(begging ~ sex + food + sex * food, data = owl)(food^2),
                random =~ 1|nest, data = owl
               )

> mod.lmer = lmer(begging ~ food + sex + food * sex + I(food^2) + (1|nest),
                  data = owl
                 )

> mod.glmer = glmer(begging ~ food + sex + food * sex + I(food^2) + (1|nest),
                    data=owl, family=poisson
                   )

> par(mfrow=c(2,2))

> plot(mod.lm)
```

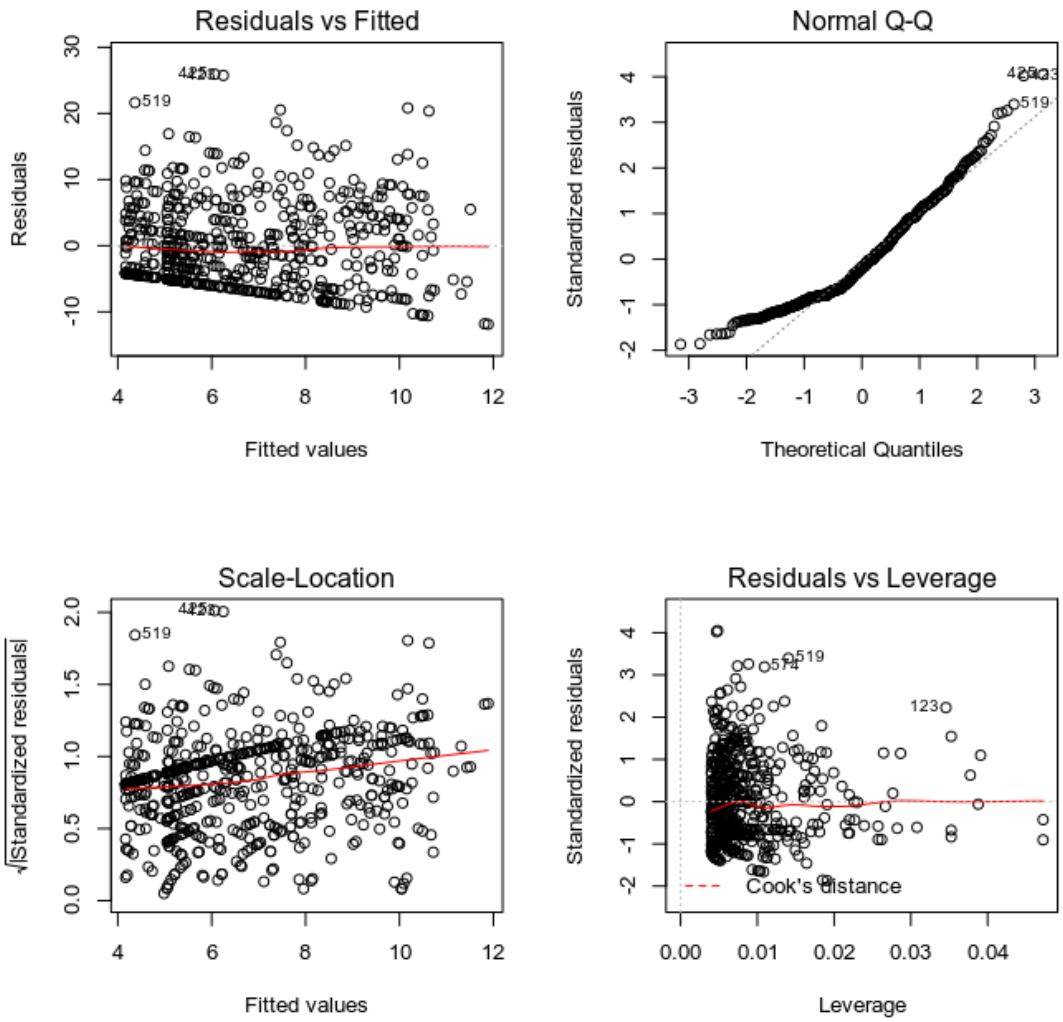


Illustration 56: Plot - Fitting Linear Models

```
> par(mfrow=c(2,2))
> plot(mod.glm)
```

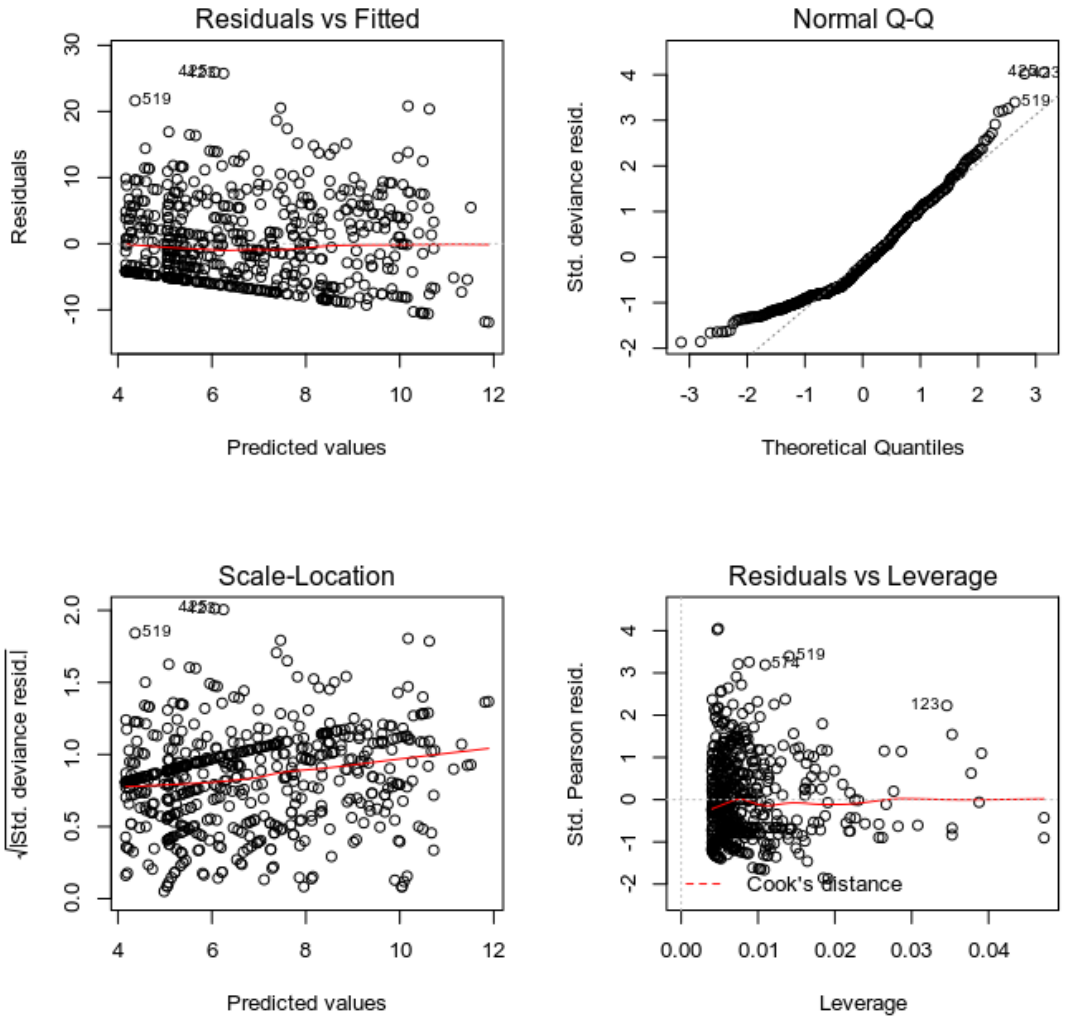


Illustration 57: Plot - Fitting Generalised Linear Models

```
> plot(mod.lme)
```

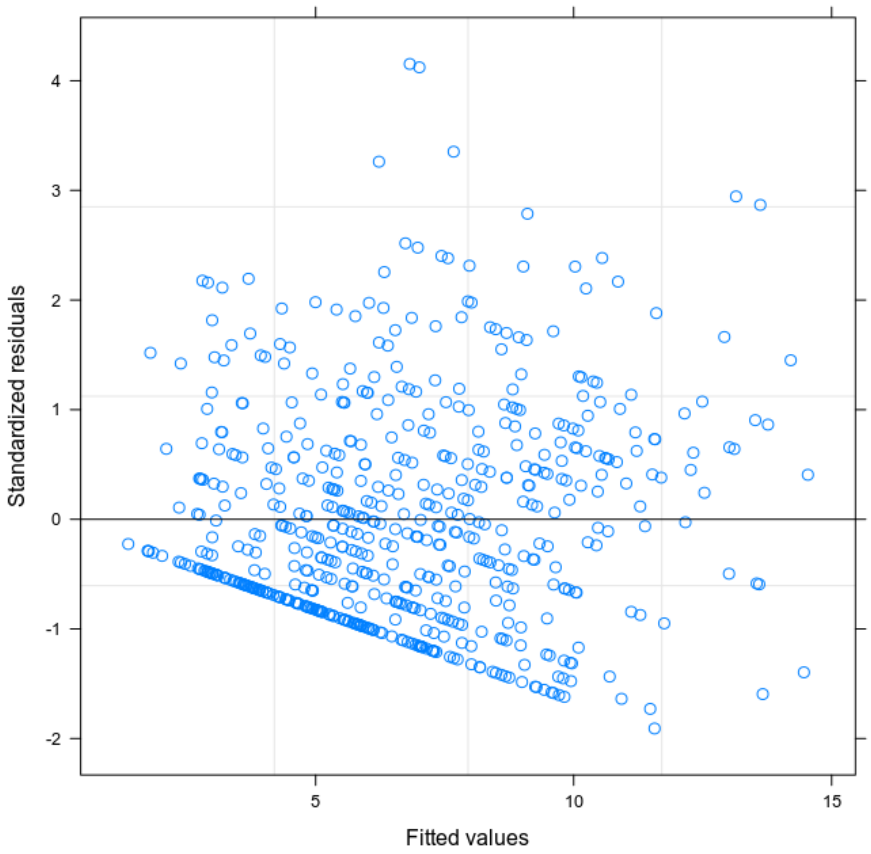


Illustration 58: Plot - Linear Mixed-Effects Models

```
> plot(mod.lmer)
```

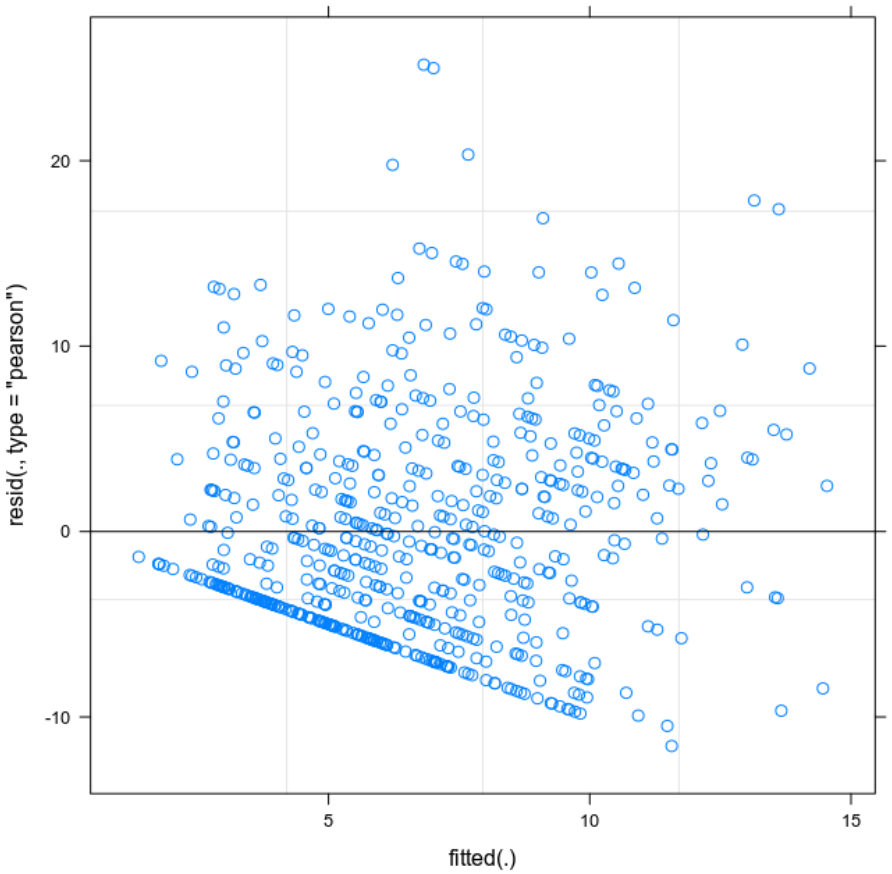


Illustration 59: Plot - Fit Linear Mixed-Effects Models

15. Qualitative Data Analysis with R

15.1 Introduction

As has already been shown the *R* programming language is very powerful for quantitative analysis, but what or Qualitative analysis? *R* has a R Qualitative Data Analysis (RQDA) for qualitative text and PDF document analysis.

It is particularly useful for inductive thematic analysis however for deductive analysis it is necessary to upload Categories and Codes one by one. RQDA Code Builder resolves this.

This document demonstrates how to use *RQDA()* and the *RQDA Code Builder* on a GNU/Linux platform. *R* and *RQDA()* can be used on other platforms like Microsoft Windows and as the RQDA Code Builder is Python3 based it can easily be adapted for other platform implementations.

It is necessary to have *python3* installed on the platform. Use the *Software Manager* for your GNU/Linux flavour or install using *apt* from the shell terminal.

```
$ sudo apt install python3
$ sudo apt-get install python-yaml
```

Confirm the install and the version of *python3*.

```
$ python3 --version
  Python 3.5.2
```

15.2 Qualitative Content Analysis

Qualitative Content Analysis follows a procedure (Flick, 2014):

1. Deciding the research question
2. Selecting material
3. Building a coding frame
4. Segmentation
5. Trial coding
6. Evaluating and modifying the coding frame
7. Main analysis
8. Presenting and interpreting the findings.

15.3 Coding

Assuming that steps 1 and 2 are completed and the next step is the building of a coding frame. There are two approaches, inductive and deductive.

The inductive approach has codes extracted directly from the source data. As the researcher reads through each source file (interviews, papers, etc..), he or she highlights key lines and creates a code for it. These codes are added and modified as the researcher reads through all the source material. The codes are then bundled into codes of common category. RDQA() is very suitable for this approach.

The deductive approach involves the researcher developing codes and categories in advance, in a scheme. These codes are then applied to the source data. As RDQA()

expects codes to be added one by one through the graphical interface this is difficult. Application of the *rqda_code_builder.py* program described here helps to fix this.

15.4 Starting RQDA()

Create a directory as a parent for the project and open a command shell in it. Within the parent directory create a *Sources* directory. Place the source files in the *Sources* directory. In this example you can see two source files but typically this would be many files associated with interviews, observation logs, etc..

```
$ mkdir Sources
```

Illustration 60: Sources directory

```
$ ls Sources
Colours_of_Health_and_Sickness_Sociocult.txt
Psychological_Properties_Of_Colours.txt
```

Run the 'R' program.

```
$ R --quiet

>
```

15.4.1 Add the RQDA library

Add the *RDQA() library*, this is the program that allows the researcher to analyse the data.

```
> library(RQDA)
Loading required package: RSQLite
Loading required package: gWidgetsRGtk2
Loading required package: RGtk2
Loading required package: gWidgets
Loading required package: cairoDevice
Loading required package: DBI

Use 'RQDA()' to start the programme.
```

The graphical tool starts.

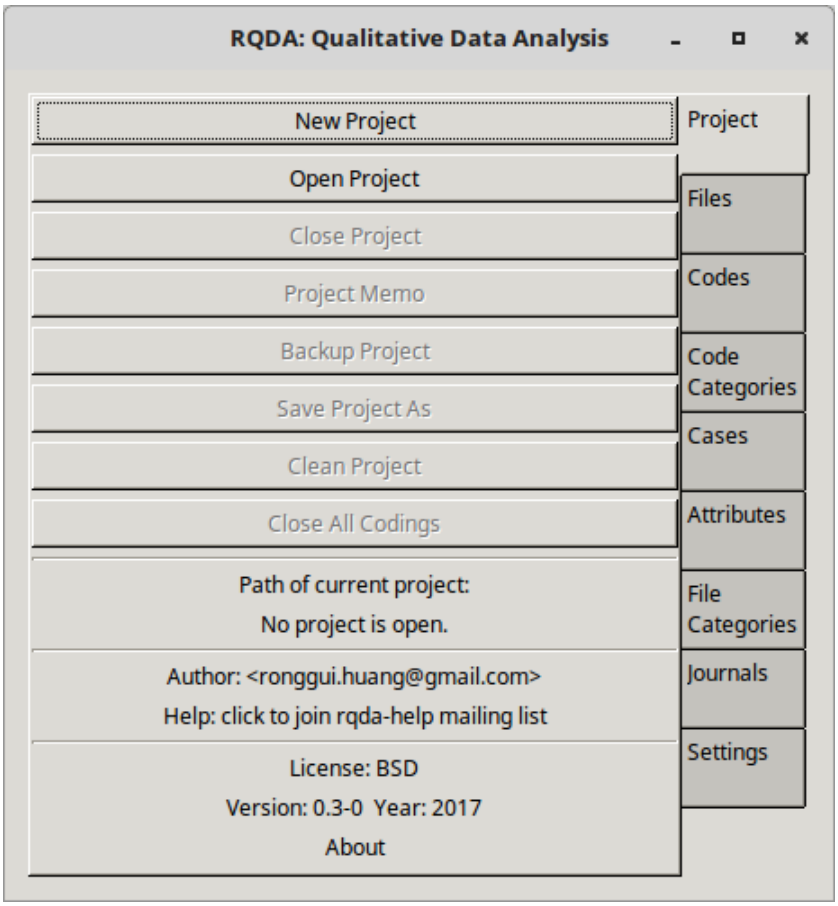


Illustration 61: RQDA() GUI

15.4.2 Create a Project

In the Graphical User Interface (GUI):

- Click *New Project*.
- Enter a name in the desired path: *Colour_project.rqda*. - Click *OK*.

A new project file appears in the directory.

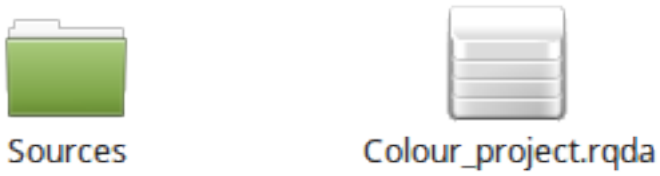


Illustration 62: Project SQLite database

You may also notice in the *R* shell that the following command is executed.

```
> [1] "~/Colour_project.rqda"
```

Name the coder

Select the *Settings* tab and define the *Name of Coder* in the first box.

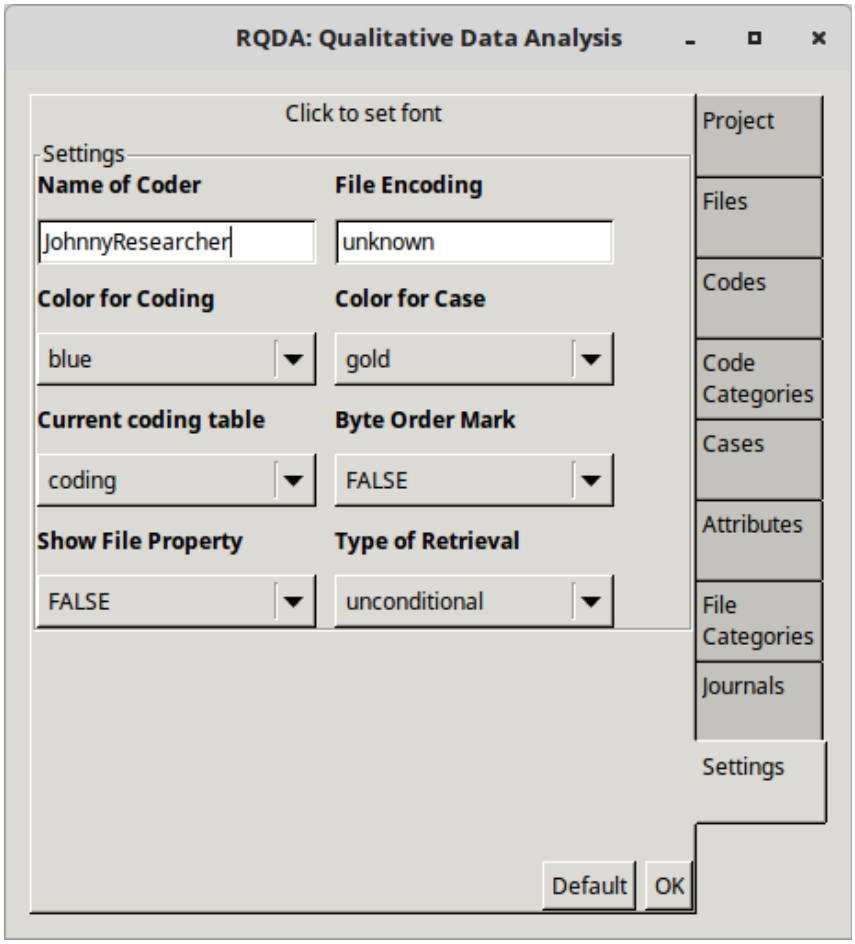


Illustration 63: RQDA() interface

15.4.3 Import source files to the project

The next step is to import source data. This can be achieved either through the GUI one by one, or in bulk using the *R* function *write.FileList()* in the *R* shell.

Using the GUI

To use the GUI, select:

- The *Files* tab followed by the *Import* button.
- Browse to each file in turn and select.

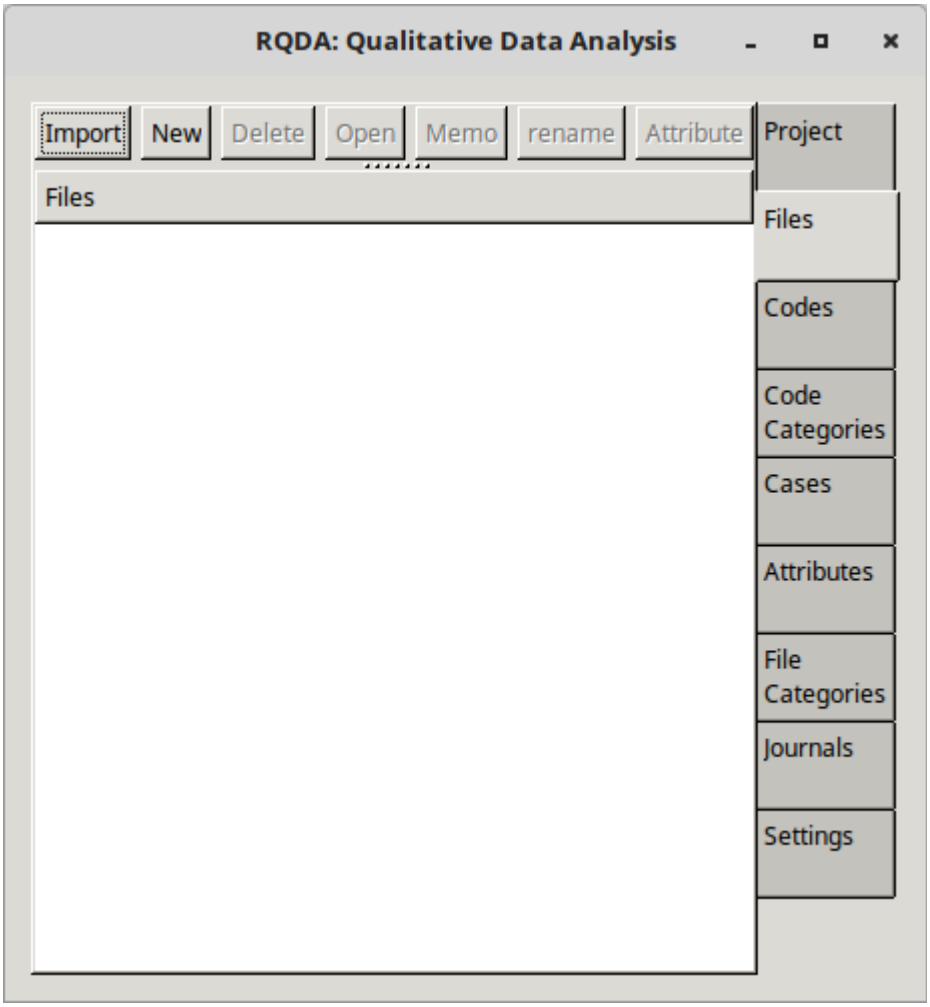


Illustration 64: RQDA() files

Using the R shell

An alternative mechanism is to use the *R* shell. This command using the *addFilesFromDir()* function selects the files in the *Sources* directory that match the pattern. In this case all files that end in the pattern *.txt*.

Execute the command:

```
> addFilesFromDir('Sources', pattern = "*.txt$")
```

If you now check the GUI by clicking the *Files* tab, you will notice that the files from the *Sources* directory have been imported. Alternatively use the *getFiles()* function in the *R* shell to confirm.

```
> getFiles()
[1] "Colours_of_Health_and_Sickness_Sociocult.txt"
[2] "Psychological_Properties_Of_Colours.txt"
attr(,"class")
[1] "RQDA.vector" "fileName"
```

15.5 Coding

15.5.1 Inductive approach

RQDA() is very suitable for the *inductive approach* however it takes significant time.

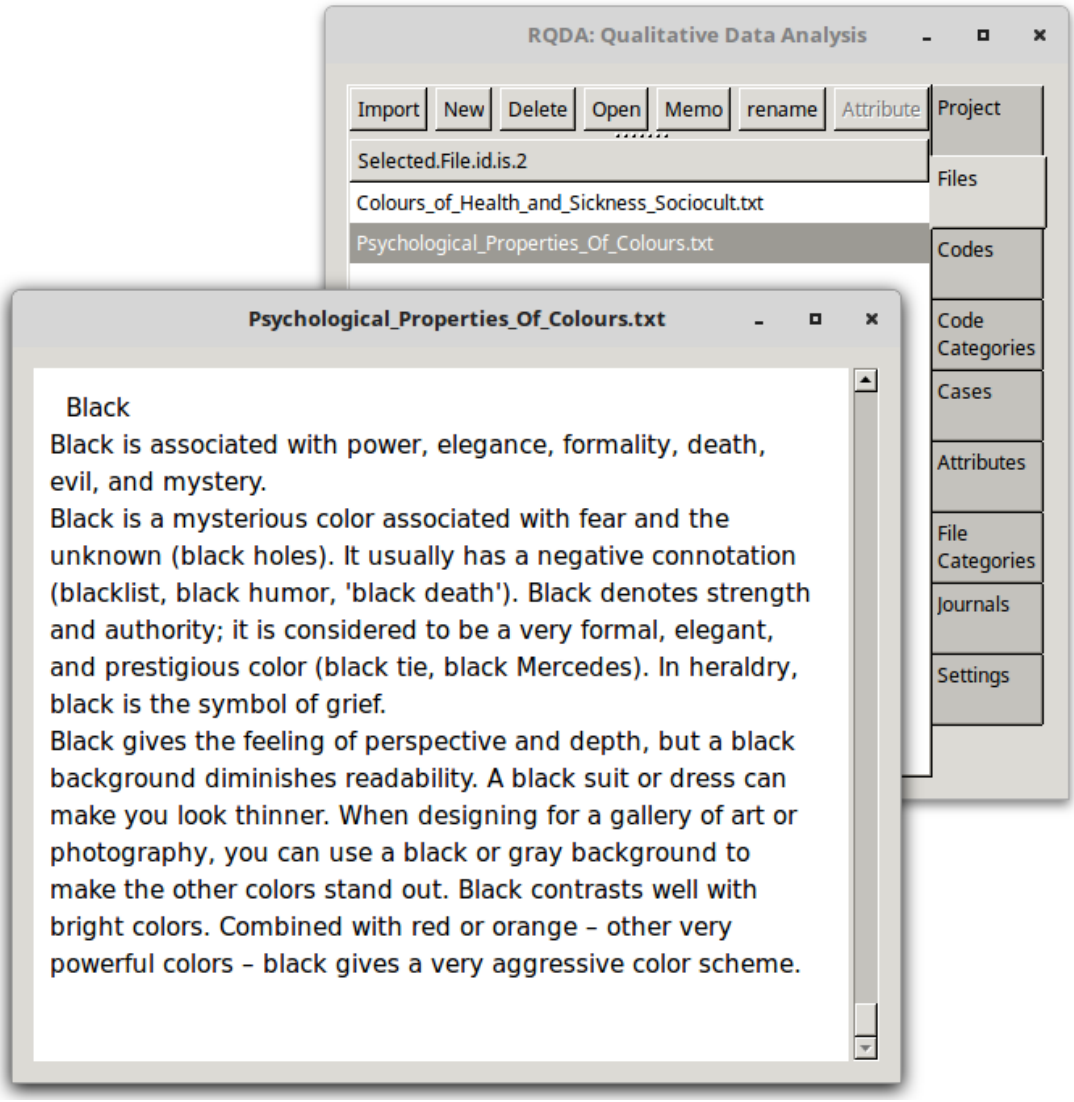


Illustration 65: RQDA() files 2

Select each document in turn from the *Files* tab, a popup appears with the text from the source file selected.

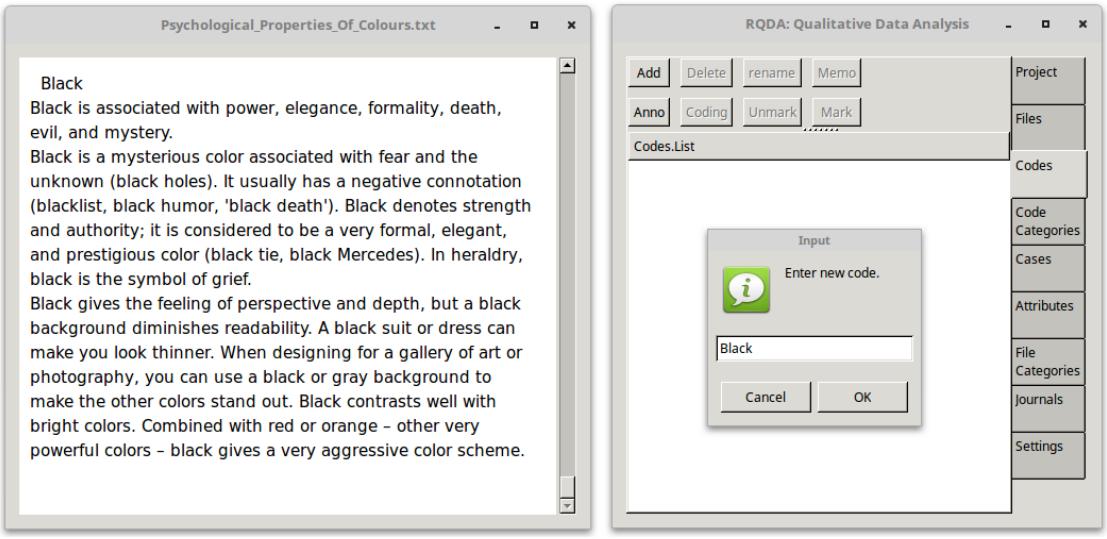


Illustration 66: RQDA() Codes

On the main GUI click the *Codes* tab and as a line is read that requires coding select *Add* and create the code. For example, to add a code *Black*, click *Add*. Enter the new code in the box provided and click *OK*. With text highlighted, select the appropriate code, i.e. *Black* and click *Mark*.

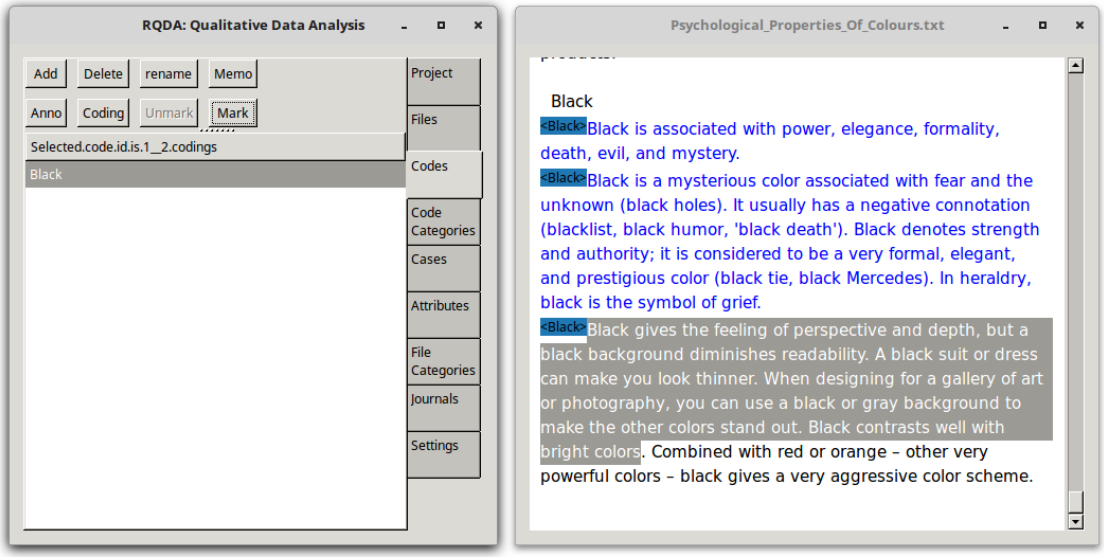


Illustration 67: RQDA() Codes 2

As can be seen each line is tagged.

15.5.2 Deductive approach

Unfortunately there does not appear to be a mechanism to import codes into the *RQDA() database* in bulk. For the *deductive approach* a researcher may have tens or even hundreds of categories and codes, it could be necessary to bulk upload. Extract the files from the *RQDA-Code-Builder.tgz* archive file which will give all the files including the database from this example as well as the *rqda_code_builder.py*. Move this file to the parent directory of the *Sources* directory.

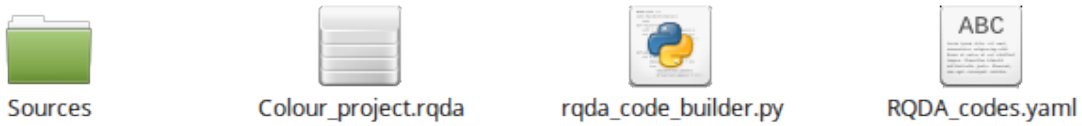


Illustration 68: RQDA() Code Builder

The RQDA Code Builder (*rqda_code_builder.py*) program resolves this.

YAML file

YAML Ain't Markup Language (YAML) is a human-readable data serialisation language that is commonly used for configuration files, but can be used in many applications where data is being stored or transmitted. It is an ideal format for mapping of categories and codes.

The example project demonstrates how to *deduct* the following code schema as a YAML file in the same directory:

```
$ cat RQDA_codes.yaml
  RQDA_codes.yaml
  ---

  Colour:
   -  'Red'
   -  'Green'
   -  'Yellow'
   -  'Grey'
   -  'Black'
   -  'White'
   -  'Black'
   -  'Blue'
   -  'Pink'
   -  'Brown'
   -  'Purple'

  Psychological Properties:
   -  'Physical'
   -  'Intellectual'
   -  'Emotional'
   -  'Balance'
   -  'Spiritual'

  Floral Metaphors:
   -  'Daisy'
   -  'Juicy'
   -  'Apple'
   -  'Berry'
   -  'Flower'
   -  'Peach'

  Human Characteristics:
   -  'Divinity'
   -  'Eternity'
   -  'Infinity'
```

Executing the RQDA Code Builder

Before executing the RQDA Code Builder it is important to shut down the *RQDA() application* by clicking on the *X* in the top right corner and selecting *OK* to the *Really EXIT?* question.

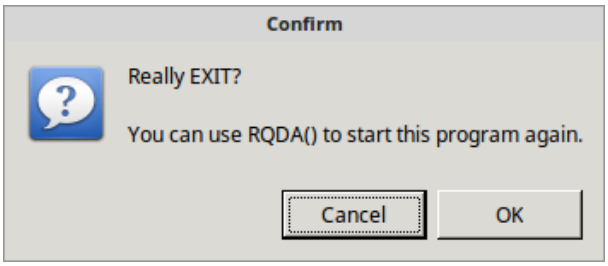


Illustration 69: RQDA() Exit

The file that the *RQDA() program* uses to store data is an SQLite database. It is the file that was created at the beginning when the project was opened (*Colour_project.rqda*). The RQDA Code Builder reads the YAML formatted schema and uploads it to the database. It also creates a Structured Query Language (SQL) log of each SQL command it executes and more importantly develops a set of *R* commands that match text blocks to the codes. It has the following switches:

```
-c|--coder [Name]   - Define coder, must match that from RQDA() settings.

-d|--database [DB] -  Define path to SQLite3 database file.

-y|--yaml [YAML]   -  Define path to YAML code file.
```

Execute the command, check it is version 1.4 or greater and execute with the relevant switches as demonstrated.

```
$ cat rqda_code_builder.py | grep '# Version' | awk {'print $4'}
  1.4

$ ./rqda_code_builder.py -c JohnnyResearcher -d Colour_project.rqda -y RQDA_codes.yaml


RQDA Code Builder
-----------------

Connecting to the SQLite3 database Colour_project.rqda.
Connected to the SQLite3 database Colour_project.rqda. Uploading..

Upload completed
----------------

A full list of SDL commands executed can be seen in the 'RQDA_SQL.log' file.
You can restart the RQDA() library with the following command in the R shell:

> RQDA()
```

Restart *RQDA()* as instructed.

```
> RQDA()
```

Two new files are created, *RQDA_SQL.log* which is a log of the SQL commands executed on the database as well as *RQDA_R_search_cmds.R* which is a list of commands that will be executed in the *R* shell to apply the deductive codes to the source files.

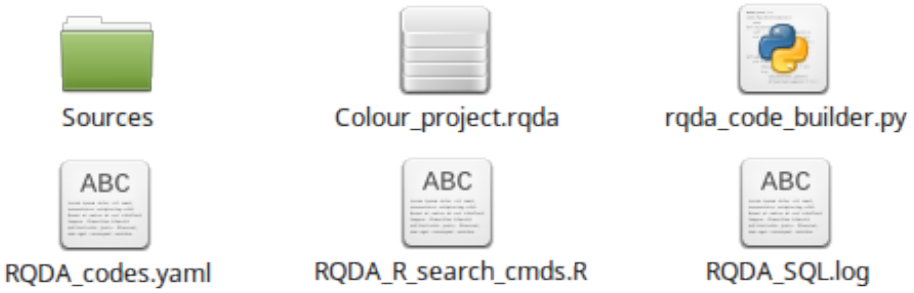


Illustration 70: RQDA() Search commands and Log files

Applying the RQDA 'R' search commands

To apply the RQDA search commands execute the following command in the *R* shell. This bulk executes the commands in the *RQDA_R_search_cmds.R* file on all the source files.

```
> source('RQDA_R_search_cmds.R')
```

15.6 Reviewing Coding

Before diving into the coding within the various source files, review the coding statistics. It can be seen from this extract that 385 code blocks were applied to the source texts.

```
> getCodingTable()
    rowid cid fid     codename                                    filename
1       4   1   2     Physical      Psychological_Properties_Of_Colours.txt
2       5   1   2     Physical      Psychological_Properties_Of_Colours.txt
3       6   1   2     Physical      Psychological_Properties_Of_Colours.txt
4       7   1   2     Physical      Psychological_Properties_Of_Colours.txt
5       8   1   2     Physical      Psychological_Properties_Of_Colours.txt
6       9   1   2     Physical      Psychological_Properties_Of_Colours.txt

...   ...  ..   .     ...       ...

382   385  24   1     Eternity Colours_of_Health_and_Sickness_Sociocult.txt
383   386  24   1     Eternity Colours_of_Health_and_Sickness_Sociocult.txt
384   387  25   1     Infinity Colours_of_Health_and_Sickness_Sociocult.txt
385   388  25   1     Infinity Colours_of_Health_and_Sickness_Sociocult.txt
    index1 index2 CodingLength
1      399    534          135
2     4109   4197           88
3     4739   4842          103
4      848    977          129
5     1454   1587          133
6     4257   4394          137

...   ...     ...          ...

382   3189   3301          112
383  15902  16020          118
384   3189   3301          112
385  15902  16020          118
```

15.7 Reviewing the coded blocks

Selecting the *Codes* tab from the *RQDA()* GUI and select any particular code. In this case *Brown* was selected. Click on the *Coding* button and a popup appears with each instance of sentences within the source files where the word *Brown* or *brown* appeared, such sentences were tagged with the *Brown* tag. The popup also shows for each block the source file from where the sentence appeared.

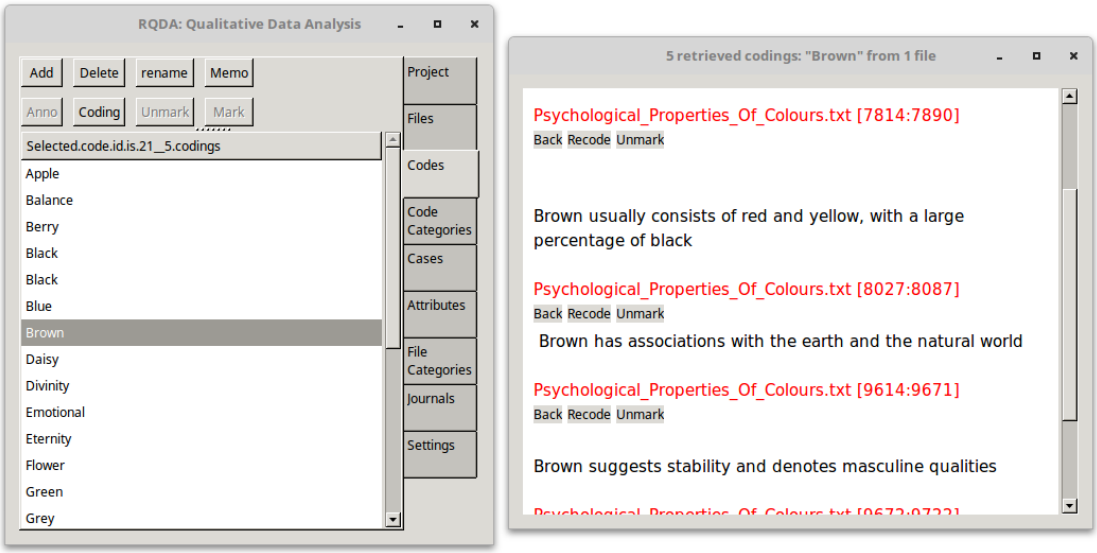


Illustration 71: Reviewing the coded blocks

This performs an initial *deductive coding*. There may be quirks however, what if one interviewee kept referring to *Beige* but the researcher wanted to code it as *Brown*? or the researcher has a code *Colour* and some of the transcripts were transcribed in American English. In this case sentences with *Color* should be coded with *Colour*.

Carry out additional coding of sentences like this.

First find the *CID* of the *Code* for *Brown*. Select the *Codes* tab, click on the *Brown* code and its *CID* can be seen at the top of the pane as shown by the red circle in Illustration 72.

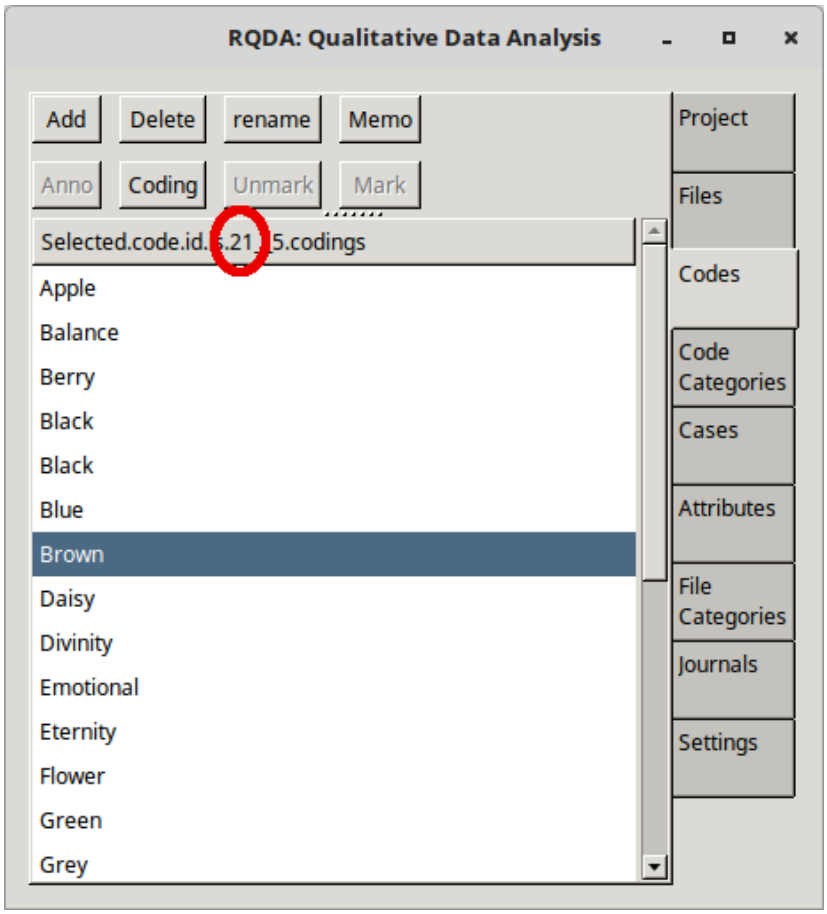


Illustration 72: Find the CID of a code

Execute the following two lines in the *R* shell and they will be added to the main coding already performed.

```
> codingBySearch("Beige",fid=getFileIds(),cid=21,seperator="[.!?]")

> codingBySearch("beige",fid=getFileIds(),cid=21,seperator="[.!?]")
```

15.8 Visualising categories

There are some tools built into *RQDA()* for visualisation. For example using the *D3.js JavaScript library* for manipulating data. *D3* helps bring data to life visually using Hypertext Markup Language (HTML), Scalable Vector Graphics (SVG), and Cascading Style Sheets (CSS).

15.8.1 Installing d3Network

On the *R* Shell install D3.js and activate the *d3Network* within *R*.

```
> install.packages('d3Network')

> library(d3Network)
```

15.8.2 Visualising Categories

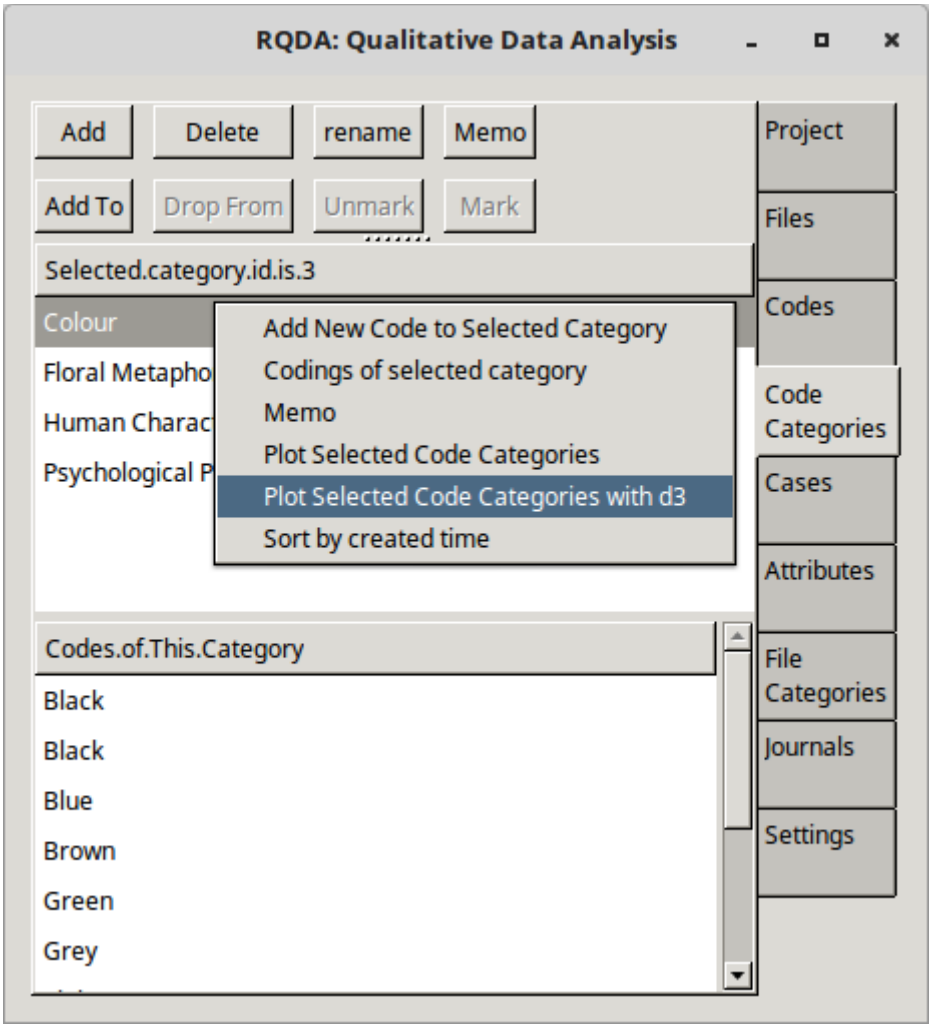


Illustration 73: Plot selected code categories with d3

Select the *Categories* tab, highlight a *Category* or many *Categories* using the *ctrl* button and right click. Scroll down to the *Plot selected code categories with d3*. A HTML page will pop-up with diagrams like these:

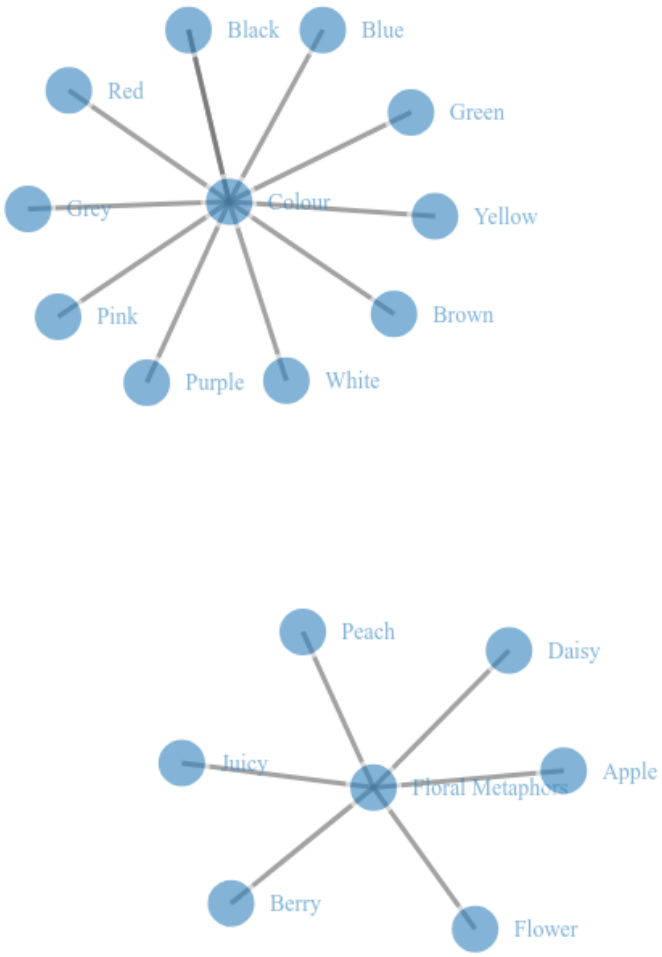


Illustration 74: d3 plot

15.9 Summary

There are a lot more features to *R* and *RQDA()* that can aid qualitative research. The additional *RQDA Code Builder* program (*rqda_code_builder.py*) allows the researcher to deductively pre-build a code schema and apply it automatically.

16. Bibliography

Bates, Douglas,. Maechler, Martin., Bolker, Ben., Walker, Steve. (2015). Fitting Linear Mixed-Effects Models Using lme4. Journal of Statistical Software, 67(1), 1-48. doi:10.18637/jss.v067.i01.

Crawley, J. Michael. (2012). The R Book, Second Edition. Wiley. 7 Nov 2012. ISBN: 978-0-470-97392-9.

Dalgaard, Peter. (2008). Introductory Statistics with R, Second Edition. Springer. 15 Aug 2008 ISBN-13: 978-0-387-79053-4.

Gandrud, Christopher. (2015). d3Network: Tools for creating D3 JavaScript network, tree, dendrogram, and Sankey graphs from R[online]. Available at: https://CRAN.R-project.org/package=d3Network. R package version 0.5.2.1.

Huang, Ronggui. (2018). RQDA: R-based Qualitative Data Analysis [online]. Available at: http://rqda.r-forge.r-project.org. R package version 0.3-1.

Low, Matt and Hiron, Matthew. (2018). Programming & Statistics in the R Computer Language (Course slides). Department of Ecology, Swedish University of Agricultural Sciences, Uppsala, Sweden, Sep 2018.

Pinheiro J, Bates D, DebRoy S, Sarkar D and R Core Team (2018). nlme: Linear and Nonlinear Mixed Effects Models [online]. Available at: https://CRAN.R-project.org/package=nlme. R package version 3.1-137.

R Core Team (2018). R: A language and environment for statistical computing. R Foundation for Statistical Computing [online]. Available: https://www.R-project.org. Vienna, Austria.

Short, Tom. (2011). R Reference Card. EPRI PEAC, 07 Nov 2011.

Teetor, Paul. (2011). R Cookbook. O'Reilly Media Inc. Mar 2011. ISBN: 978-0-596-80915-7. March 2011.

Uwe Flick. (2014). An Introduction to Qualitative Research. 5th Edition. ISBN: 978-1-4462-6778-3. Sage Publications Ltd. 20 Jan 2014.

Wickham, Hadley (2018). scales: Scale Functions for Visualisation [online]. Available at: https://CRAN.R-project.org/package=scales. R package version 1.0.0.

17. GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary

then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license

notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organisation as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been

terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorises you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organisation.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.