Practical R for Epidemiologists

Mark Myatt

2018-04-21

Contents

Chapter 1

Introduction

1.1 Introducing R

R is a system for data manipulation, calculation, and graphics. It provides:

- Facilities for data handling and storage
- A large collection of tools for data analysis
- Graphical facilities for data analysis and display
- A simple but powerful programming language

R is often described as an environment for working with data. This is in contrast to a *package* which is a collection of very specific tools. R is not strictly a statistics system but a system that provides many classical and modern statistical procedures as part of a broader data-analysis tool. This is an important difference between R and other statistical systems. In R a statistical analysis is usually performed as a series of steps with intermediate results being stored in objects. Systems such as SPSS and SAS provide copious output from (e.g.) a regression analysis whereas R will give minimal output and store the results of a fit for subsequent interrogation or use with other R functions. This means that R can be tailored to produce exactly the analysis and results that you want rather than produce an analysis designed to fit all situations.

R is a language based product. This means that you interact with R by typing commands such as:

```
table(SEX, LIFE)
```

rather than by using menus, dialog boxes, selection lists, and buttons. This may seem to be a drawback but it means that the system is considerably more flexible than one that relies on menus, buttons, and boxes. It also means that every stage of your data management and analysis can be recorded and edited and re-run at a later date. It also provides an audit trail for quality control purposes.

R is available under UNIX (including Linux), the Macintosh operating system OS X, and Microsoft Windows. The method used for starting R will vary from system to system. On UNIX systems you may need to issue the R command in a terminal session or click on an icon or menu option if your system has a windowing system. On Macintosh systems R will be available as an application but can also be run in a terminal session. On Microsoft Windows systems there will usually be an icon on the Start menu or the desktop.

R is an open source system and is available under the *GNU general public license* (GPL) which means that it is available for free but that there are some restrictions on how you are allowed to distribute the system and how you may charge for bespoke data analysis solutions written using the R system. Details of the general public license are available from http://www.gnu.org/copyleft/gpl.html.

R is available for download from http://www.r-project.org/.

This is also the best place to get extension packages and documentation. You may also subscribe to the R mailing lists from this site. R is supported through mailing lists. The level of support is at least as good as for commercial packages. It is typical to have queries answered in a matter of a few hours.

Even though R is a free package it is more powerful than most commercial packages. Many of the modern procedures found in commercial packages were first developed and tested using R or **S-Plus** (the commercial equivalent of R).

When you start R it will issue a prompt when it expects user input. The default prompt is:

```
>
```

This is where you type commands that call functions that instruct R to (e.g.) read a data file, recode data, produce a table, or fit a regression. For example:

```
> table(SEX, LIFE)
```

If a command you type is not complete then the prompt will change to:

```
+
```

on subsequent lines until the command is complete:

```
> table(
+ SEX, LIFE +)
```

The > and + prompts are not shown in the example commands in the rest of this material.

The example commands in this material are often broken into shorter lines and indented for ease of understanding. The code still works as lines are split in places where R knows that a line is not complete. For example:

```
table(SEX,
      LIFE)
```

could be entered on a single line as:

```
table(SEX, LIFE)
```

In this example R knows that the command is not complete until the brackets are closed. The following example could also be written on one line:

```
salex.lreg.coeffs <-
  coef(summary(salex.lreg))
```

In this case R knows that the <- operator at the end of the first line needs further input.

R maintains a history of previous commands. These can be recalled and edited using the up and down arrow keys.

Output that has scrolled off the top of the output / command window can be recalled using the window or terminal scroll bars.

Output can be saved using the `sink()` function with a file name: `sink("results.out")` to start recording output. Use the `sink()` function without a file name to stop recording output: `sink()`

You can also use clipboard functions such as copy and paste to (e.g.) copy and then paste selected chunks of output into an editor or word processor running alongside R.

All the sample data files used in the exercises in this manual are space delimited text files using the general format:

```
ID AGE IQ
1 39 94
2 41 89
```

```
3 42 83
4 30 99
5 35 94
6 44 90
7 31 94
8 39 87
```

R has facilities for working with files in different formats including (through the use of extension packages) **ODBC** (open database connectivity) and **SQL** data sources, **EpiInfo**, **EpiData**, **Minitab**, **SPSS**, **SAS**, **S-Plus**, and **Stata** format files.

1.2 Retrieving data

All of the exercises in this manual assume that the necessary data files are located in the current working directory. All of the data files that you require to follow this material are in a ZIP archive that can be downloaded from:

http://www.brixtonhealth.com/prfe/prfe.zip

A command such as:

```
read.table("data/fem.dat", header = TRUE)
```

retrieves the data stored in the file named `fem.dat` which is stored in the `data` folder.

To retrieve data that is stored in files outside a different directory you need to specify the full path to the file. For example:

```
read.table("~/example/fem.dat", header = TRUE)
```

will retrieve the data stored in the file named `fem.dat` stored in the `example` directory under the user's home directory on UNIX, Linux, and OS X systems.

R follows many UNIX operating and naming conventions including the use of the backslash (\) character to specify special characters in strings (e.g. using `\n` to specify a new line in printed output). Windows uses the backslash (\) character to separate directory and file names in paths. This means that Windows users need to escape any backslashes in file paths using an additional backslash character. For example:

```
read.table("c:\\example\\fem.dat", header = TRUE)
```

will retrieve the data that is stored in the file named `fem.dat` which is stored in the `example` directory off the root directory of the `C:` drive. The Windows version of R also allows you to specify UNIX-style path names (i.e. using the forward slash (`/`) character as a separator in file paths). For example:

```
read.table("c:/example/fem.dat", header = TRUE)
```

Path names may include shortcut characters such as:

`.`	The current working directory
`..`	Up one level in the directory tree
`~`	The user's home directory (on UNIX-based systems)

R also allows you to retrieve files from any location that may be represented by a standard `uniform resource locator (URL)` string. For example:

```
read.table("file://~/example/fem.dat", header = TRUE)
```

will retrieve the data stored in the file named `fem.dat` stored in the `example` directory under the users home

directory on UNIX-based systems.

All of the data files used in this section are stored in the `/data` directory of this guide's GitLab repository (https://git.validmeasures.org/datahub/datahubguide/tree/master/data). This means, for example, that you can use the `read.table()` function specifying

"https://git.validmeasures.org/datahub/datahubguide/tree/master/data/fem.dat"

as the `URL` to retrieve the data that is stored in the file named `fem.dat` which is stored in the `/data` directory of this guide's GitLab repository.

Chapter 2

Getting acquainted with R

In this exercise we will use R to read a dataset and produce some descriptive statistics, produce some charts, and perform some simple statistical inference. The aim of the exercise is for you to become familiar with R and some basic R functions and objects.

The first thing we will do, after starting R, is issue a command to retrieve an example dataset:

```
fem <- read.table("fem.dat", header = TRUE)
```

This command illustrates some key things about the way R works.

We are instructing R to assign (using the <- operator) the output of the `read.table()` function to an object called `fem`.

The `fem` object will contain the data held in the file `fem.dat` as an R data.frame object:

```
class(fem)
```

```
## [1] "data.frame"
```

You can inspect the contents of the `fem` data.frame (or any other R object) just by typing its name:

```
fem
```

```
##    ID AGE IQ ANX DEP SLP SEX LIFE    WT
## 1   1  39 94   2   2   2   1    1  2.23
## 2   2  41 89   2   2   2   1    1  1.00
## 3   3  42 83   3   3   2   1    1  1.82
## 4   4  30 99   2   2   2   1    1 -1.18
## 5   5  35 94   2   1   1   1    2 -0.14
## 6   6  44 90  NA   1   2   2    2  0.41
```

Note that the `fem` object is built from other objects. These are the named vectors (columns) in the dataset:

```
names(fem)
```

```
## [1] "ID"   "AGE"  "IQ"   "ANX"  "DEP"  "SLP"  "SEX"  "LIFE" "WT"
```

The [1] displayed before the column names refers to the numbered position of the first name in the output. These positions are known as indexes and can be used to refer to individual items. For example:

```
names(fem)[1]
```

```
## [1] "ID"
```

```
names(fem)[8]
```

```
## [1] "LIFE"
```

```
names(fem)[2:4]
```

```
## [1] "AGE" "IQ"  "ANX"
```

The data consist of 118 records:

```
nrow(fem)
```

```
## [1] 118
```

each with nine variables:

```
ncol(fem)
```

```
## [1] 9
```

for female psychiatric patients.

The columns in the dataset are:

ID	Patient ID
AGE	Age in years
IQ	IQ score
ANX	Anxiety (1=none, 2=mild, 3=moderate, 4=severe)
DEP	Depression (1=none, 2=mild, 3=moderate or severe)
SLP	Sleeping normally (1=yes, 2=no)
SEX	Lost interest in sex (1=yes, 2=no)
LIFE	Considered suicide (1=yes, 2=no)
WT	Weight change (kg) in previous 6 months

The first ten records of the `fem` data.frame are:

```
##    ID AGE  IQ ANX DEP SLP SEX LIFE    WT
## 1   1  39  94   2   2   2   1    1  2.23
## 2   2  41  89   2   2   2   1    1  1.00
## 3   3  42  83   3   3   2   1    1  1.82
## 4   4  30  99   2   2   2   1    1 -1.18
## 5   5  35  94   2   1   1   1    2 -0.14
## 6   6  44  90  NA   1   2   2    2  0.41
## 7   7  31  94   2   2  NA   1    1 -0.68
## 8   8  39  87   3   2   2   1    2  1.59
## 9   9  35 -99   3   2   2   1    1 -0.55
## 10 10  33  92   2   2   2   1    1  0.36
```

You may check this by asking `R` to display all columns of the first ten records in the `fem` data.frame:

```
fem[1:10, ]
```

```
##    ID AGE  IQ ANX DEP SLP SEX LIFE    WT
## 1   1  39  94   2   2   2   1    1  2.23
## 2   2  41  89   2   2   2   1    1  1.00
## 3   3  42  83   3   3   2   1    1  1.82
## 4   4  30  99   2   2   2   1    1 -1.18
## 5   5  35  94   2   1   1   1    2 -0.14
## 6   6  44  90  NA   1   2   2    2  0.41
```

```
## 7   7  31  94   2   2  NA   1    1 -0.68
## 8   8  39  87   3   2   2   1    2  1.59
## 9   9  35 -99   3   2   2   1    1 -0.55
## 10 10  33  92   2   2   2   1    1  0.36
```

The space after the comma is optional. You can think of it as a *placeholder* for where you would specify the indexes for columns you wanted to display. For example:

```
fem[1:10,2:4]
```

displays the first ten rows and the second, third and fourth columns of the `fem` data.frame:

```
##    AGE  IQ ANX
## 1   39  94   2
## 2   41  89   2
## 3   42  83   3
## 4   30  99   2
## 5   35  94   2
## 6   44  90  NA
## 7   31  94   2
## 8   39  87   3
## 9   35 -99   3
## 10  33  92   2
```

`NA` is a special value meaning *not available* or *missing*.

You can access the contents of a single column by name:

```
fem$IQ
```

```
##   [1]  94  89  83  99  94  90  94  87 -99  92  92  94  91  86  90 -99  91
##  [18]  82  86  88  97  96  95  87 103 -99  91  87  91  89  92  84  94  92
##  [35]  96  96  86  92 102  82  92  90  92  88  98  93  90  91 -99  92  92
##  [52]  91  91  86  95  91  96 100  99  89  89  98  98 103  91  91  94  91
##  [69]  85  92  96  90  87  95  95  87  95  88  94 -99 -99  87  92  86  93
##  [86]  92 106  93  95  95  92  98  92  88  85  92  84  92  91  86  92  89
## [103] -99  96  97  92  92  98  91  91  89  94  90  96  87  86  89 -99
```

```
fem$IQ[1:10]
```

```
##  [1]  94  89  83  99  94  90  94  87 -99  92
```

The `$` sign is used to separate the name of the data.frame and the name of the column of interest. Note that `R` is case-sensitive so that `IQ` and `iq` are ***not*** the same.

You can also access rows, columns, and individual cells by specifying row and column positions. For example, the `IQ` column is the third column in the `fem` data.frame:

```
fem[ ,3]
```

```
##   [1]  94  89  83  99  94  90  94  87 -99  92  92  94  91  86  90 -99  91
##  [18]  82  86  88  97  96  95  87 103 -99  91  87  91  89  92  84  94  92
##  [35]  96  96  86  92 102  82  92  90  92  88  98  93  90  91 -99  92  92
##  [52]  91  91  86  95  91  96 100  99  89  89  98  98 103  91  91  94  91
##  [69]  85  92  96  90  87  95  95  87  95  88  94 -99 -99  87  92  86  93
##  [86]  92 106  93  95  95  92  98  92  88  85  92  84  92  91  86  92  89
## [103] -99  96  97  92  92  98  91  91  89  94  90  96  87  86  89 -99
```

```
fem[9, ]
```

```
##   ID AGE  IQ ANX DEP SLP SEX LIFE    WT
```

```
## 9  9  35 -99   3   2   2   1    1 -0.55
```

```
fem[9,3]
```

```
## [1] -99
```

There are missing values in the `IQ` column which are all coded as **-99**. Before proceeding we must set these to the special `NA` value:

```
fem$IQ[fem$IQ == -99] <- NA
```

The term inside the square brackets is also an index. This type of index is used to refer to subsets of data held in an object that meet a particular condition. In this case we are instructing `R` to set the contents of the `IQ` variable to `NA` if the contents of the `IQ` variable is **-99**.

Check that this has worked:

```
fem$IQ
```

```
##   [1]  94  89  83  99  94  90  94  87  NA  92  92  94  91  86  90  NA  91
##  [18]  82  86  88  97  96  95  87 103  NA  91  87  91  89  92  84  94  92
##  [35]  96  96  86  92 102  82  92  90  92  88  98  93  90  91  NA  92  92
##  [52]  91  91  86  95  91  96 100  99  89  89  98  98 103  91  91  94  91
##  [69]  85  92  96  90  87  95  95  87  95  88  94  NA  NA  87  92  86  93
##  [86]  92 106  93  95  95  92  98  92  88  85  92  84  92  91  86  92  89
## [103]  NA  96  97  92  92  98  91  91  89  94  90  96  87  86  89  NA
```

We can now compare the groups who have and have not considered suicide. For example:

```
by(fem$IQ, fem$LIFE, summary)
```

Look at the help for the `by()` function:

```
help(by)
```

Note that you may use `?by` as a shortcut for `help(by)`.

The `by()` function applies another function (in this case the `summary()` function) to a column in a data.frame (in this case `fem$IQ`) split by the value of another variable (in this case `fem$LIFE`).

It can be tedious to always have to specify a data.frame each time we want to use a particular variable. We can fix this problem by 'attaching' the data.frame:

```
attach(fem)
```

```
## The following objects are masked from fem (pos = 4):
##
##     AGE, ANX, DEP, ID, IQ, LIFE, SEX, SLP, WT
```

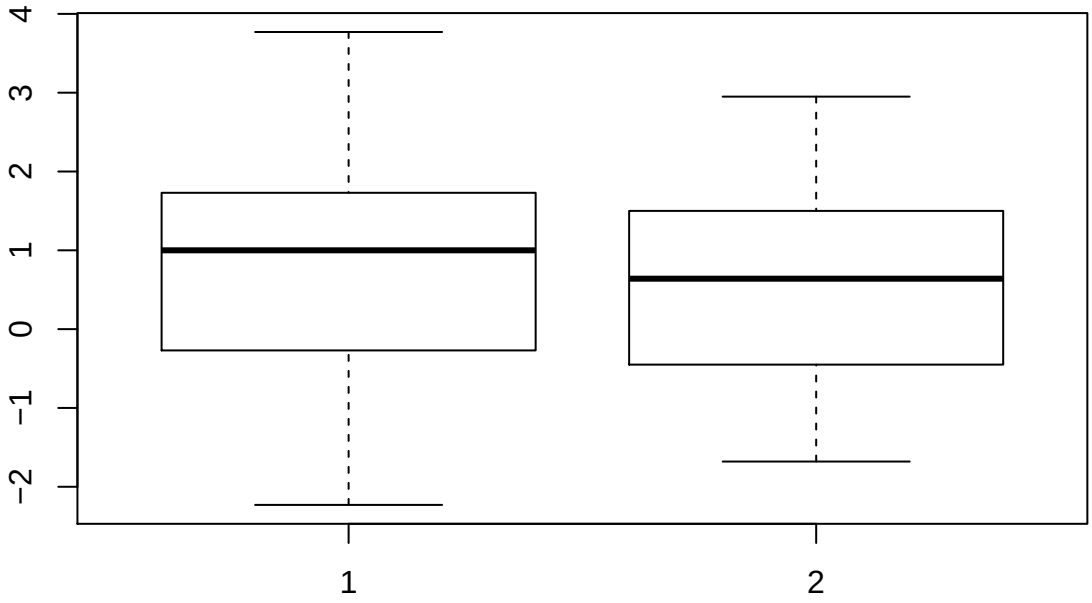
We can now refer to the columns in the `fem` data.frame without having to specify the name of the data.frame. This time we will produce summary statistics for `WT` by `LIFE`:

```
by(WT, LIFE, summary)
```

```
## LIFE: 1
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
## -2.2300 -0.2700  1.0000  0.7867  1.7300  3.7700       4
## ------------------------------------------------------------
## LIFE: 2
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
## -1.6800 -0.4500  0.6400  0.6404  1.5000  2.9500       7
```

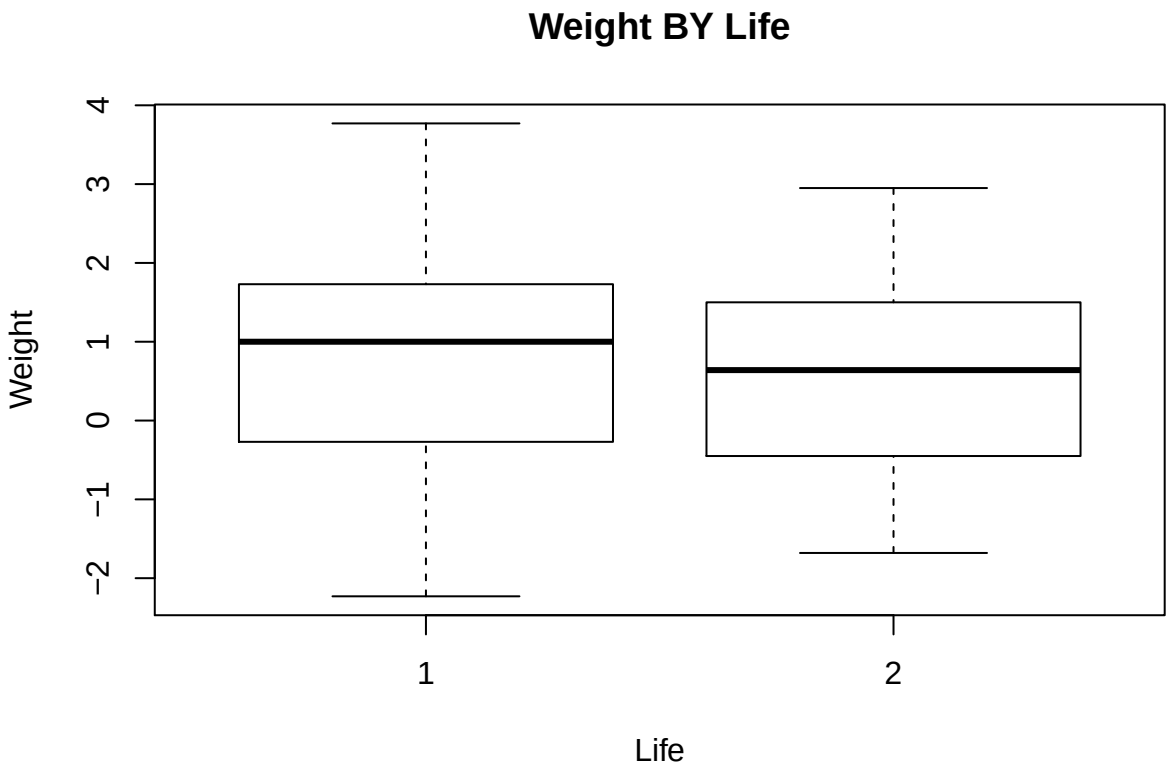
We can view the same data as a box and whisker plot:

```
boxplot(WT ~ LIFE)
```



We can add axis labels and a title to the graph:

```
boxplot(WT ~ LIFE,
        xlab = "Life",
        ylab = "Weight",
        main = "Weight BY Life")
```



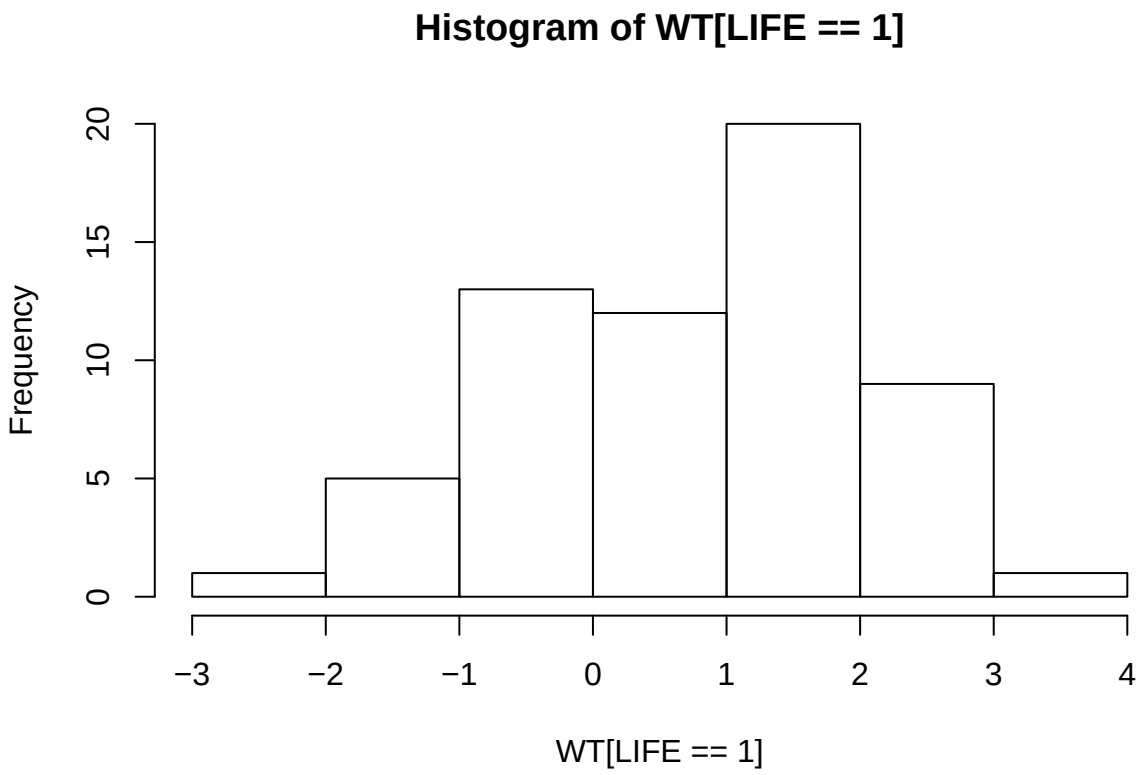
A more descriptive title might be “Weight Change BY Considered Suicide”.

The groups do not seem to differ much in their medians and the distributions appear to be reasonably

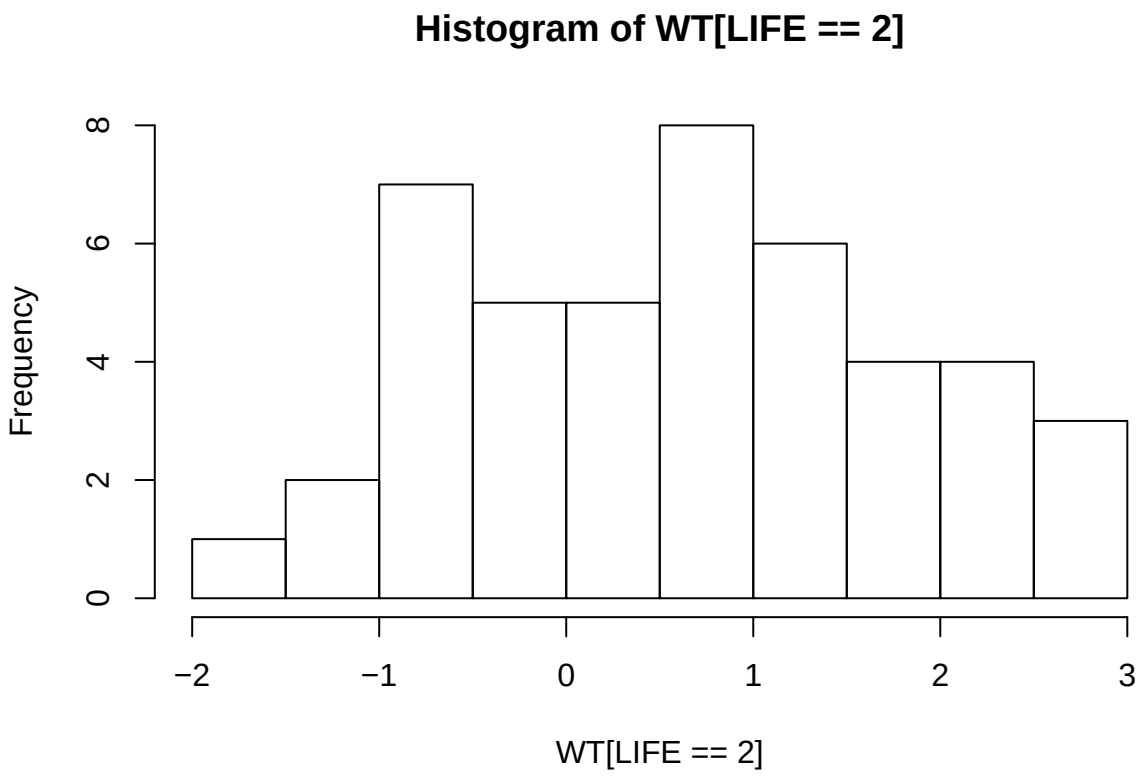
symmetrical about their medians with a similar spread of values.

We can look at the distribution as histograms:

```
hist(WT[LIFE == 1])
```

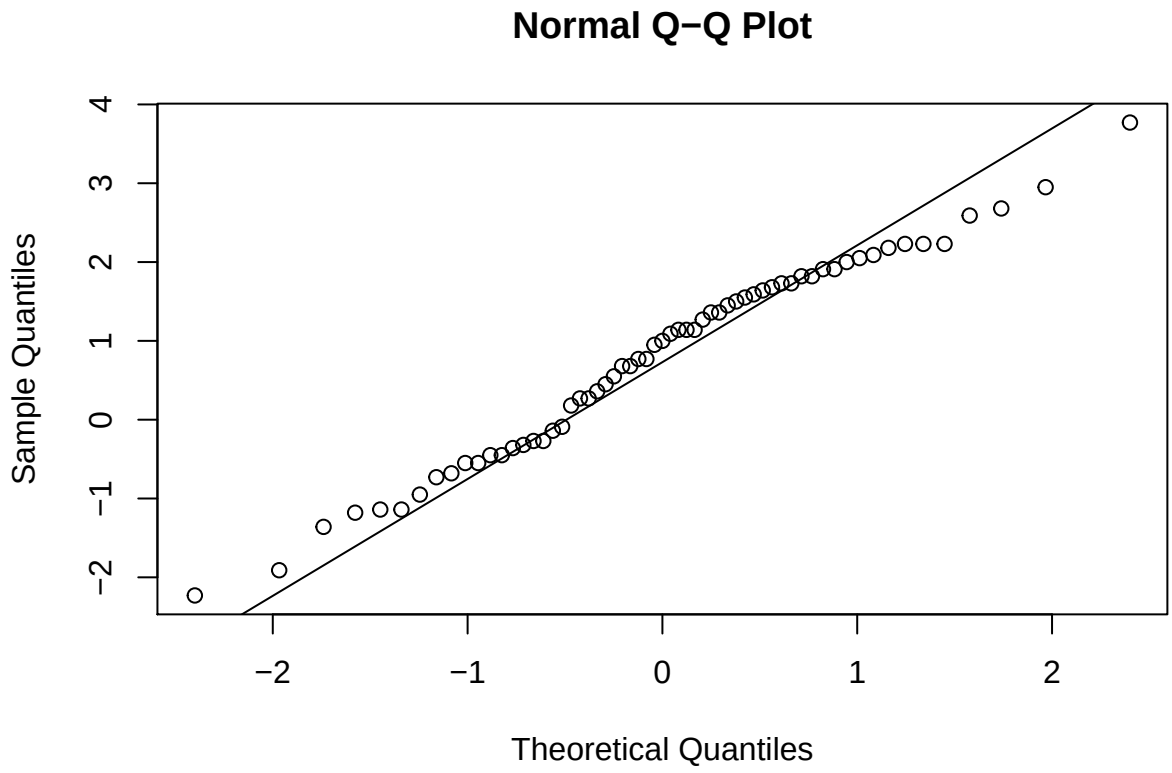



```
hist(WT[LIFE == 2])
```

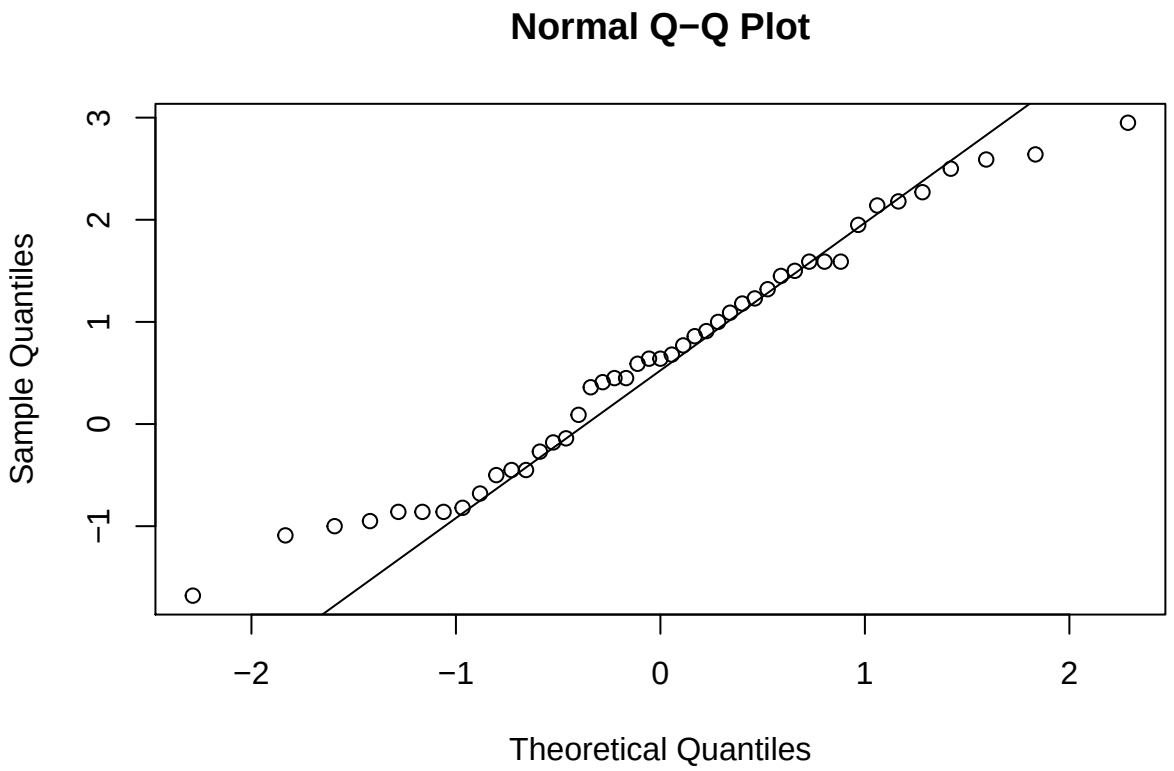


and check the assumption of normality using quantile-quantile plots:

```
qqnorm(WT[LIFE == 1])
qqline(WT[LIFE == 1])
```



```
qqnorm(WT[LIFE == 2])
qqline(WT[LIFE == 2])
```



or by using a formal test:

```
shapiro.test(WT[LIFE == 1])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  WT[LIFE == 1]
## W = 0.98038, p-value = 0.4336
```

```
shapiro.test(WT[LIFE == 2])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  WT[LIFE == 2]
## W = 0.97155, p-value = 0.3292
```

Remember that we can use the `by()` function to apply a function to a data.frame, including statistical functions such as `shapiro.test()`:

```
by(WT, LIFE, shapiro.test)
```

```
## LIFE: 1
##
##  Shapiro-Wilk normality test
##
## data:  dd[x, ]
## W = 0.98038, p-value = 0.4336
```

```
##
## ------------------------------------------------------------
## LIFE: 2
##
##  Shapiro-Wilk normality test
##
## data:  dd[x, ]
## W = 0.97155, p-value = 0.3292
```

We can also test whether the variances differ significantly using *Bartlett's test* for the homogeneity of variances:

```
bartlett.test(WT, LIFE)
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  WT and LIFE
## Bartlett's K-squared = 0.32408, df = 1, p-value = 0.5692
```

There is no significant difference between the two variances.

Many functions in `R` have a *formula interface* that may be used to specify multiple variables and the relations between multiple variables. We could have used the formula interface with the `bartlett.test()` function:

```
bartlett.test(WT ~ LIFE)
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  WT by LIFE
## Bartlett's K-squared = 0.32408, df = 1, p-value = 0.5692
```

Having checked the normality and homogeneity of variance assumptions we can proceed to carry out a `t-test`:

```
t.test(WT ~ LIFE, var.equal = TRUE)
```

```
##
##  Two Sample t-test
##
## data:  WT by LIFE
## t = 0.59869, df = 104, p-value = 0.5507
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.3382365  0.6307902
## sample estimates:
## mean in group 1 mean in group 2
##       0.7867213       0.6404444
```

There is no evidence that the two groups differ in weight change in the previous six months.

We could still have performed a `t-test` if the variances were not homogenous by setting the **var.equal** parameter of the `t.test()` function to **FALSE**:

```
t.test(WT ~ LIFE, var.equal = FALSE)
```

```
##
##  Welch Two Sample t-test
##
```

```
## data:  WT by LIFE
## t = 0.60608, df = 98.866, p-value = 0.5459
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.3326225  0.6251763
## sample estimates:
## mean in group 1 mean in group 2
##       0.7867213       0.6404444
```

or performed a non-parametric test:

```
wilcox.test(WT ~ LIFE)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  WT by LIFE
## W = 1488, p-value = 0.4622
## alternative hypothesis: true location shift is not equal to 0
```

An alternative, and more general, non-parametric test is:

```
kruskal.test(WT ~ LIFE)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  WT by LIFE
## Kruskal-Wallis chi-squared = 0.54521, df = 1, p-value = 0.4603
```

We can use the `table()` function to examine the differences in depression between the two groups:

```
table(DEP, LIFE)
```

```
##    LIFE
## DEP  1  2
##   1  0 26
##   2 42 24
##   3 16  1
```

The two distributions look very different from each other. We can test this using a chi-square test on the table:

```
chisq.test(table(DEP, LIFE))
```

```
##
##  Pearson's Chi-squared test
##
## data:  table(DEP, LIFE)
## X-squared = 43.876, df = 2, p-value = 2.968e-10
```

Note that we passed the output of the `table()` function directly to the `chisq.test()` function. We could have saved the table as an object first and then passed the object to the `chisq.test()` function:

```
tab <- table(DEP, LIFE)
chisq.test(tab)
```

```
##
##  Pearson's Chi-squared test
##
```

```
## data:  tab
## X-squared = 43.876, df = 2, p-value = 2.968e-10
```

The `tab` object contains the output of the `table()` function:

```
class(tab)
```

```
## [1] "table"
```

```
tab
```

```
##    LIFE
## DEP  1  2
##   1  0 26
##   2 42 24
##   3 16  1
```

We can pass this table object to another function. For example:

```
fisher.test(tab)
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  tab
## p-value = 1.316e-12
## alternative hypothesis: two.sided
```

When we are finished with the tab object we can delete it using the `rm()` function:

```
rm(tab)
```

You can see a list of available objects using the `ls()` function:

```
ls()
```

```
## [1] "fem"
```

This should just show the `fem` object.

We can examine the association between loss of interest in sex and considering suicide in the same way:

```
tab <- table(SEX, LIFE)
tab
```

```
##    LIFE
## SEX  1  2
##   1 58 38
##   2  5 12
```

```
fisher.test(tab)
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  tab
## p-value = 0.03175
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##   1.080298 14.214482
## sample estimates:
## odds ratio
##   3.620646
```

Note that with a two-by-two table the `fisher.test()` function produces an estimate of, and confidence intervals for, the odds ratio. Again, we will delete the `tab` object:

```
rm(tab)
```

We could have performed the Fisher exact test without creating the tab object by passing the output of the `table()` function directly to the `fisher.test()` function:

```
fisher.test(table(SEX, LIFE))
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  table(SEX, LIFE)
## p-value = 0.03175
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##   1.080298 14.214482
## sample estimates:
## odds ratio
##   3.620646
```

Choose whichever method you find easiest but remember that it is easy to save the results of any function for later use.

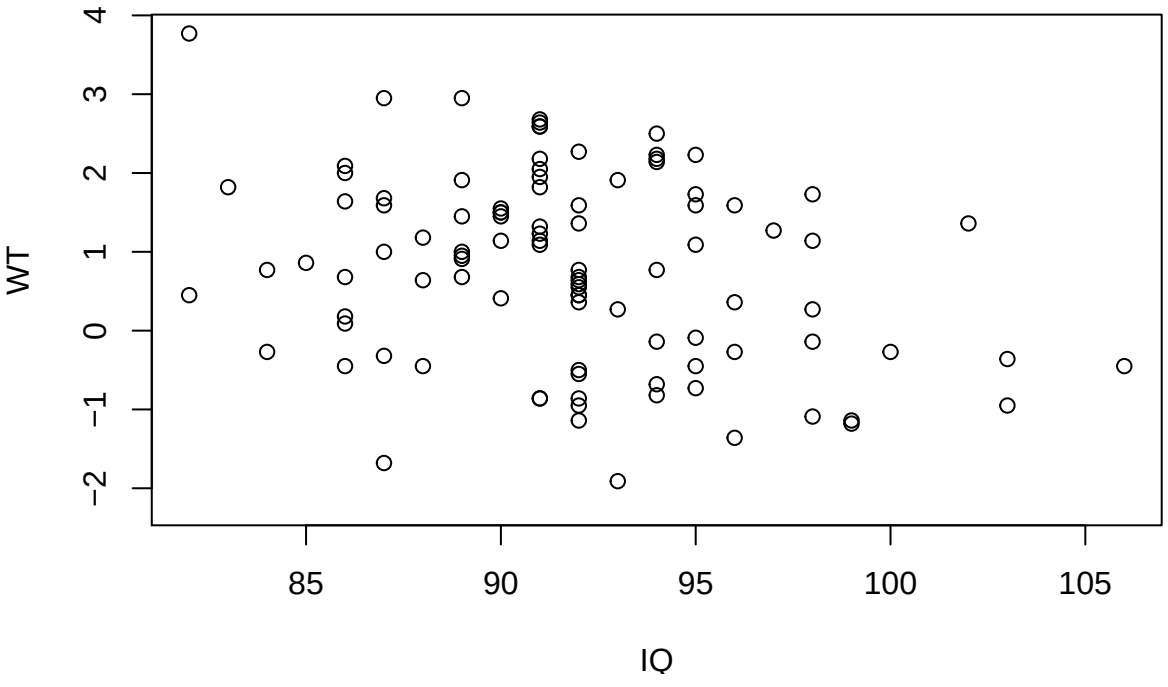
We can explore the correlation between two variables using the `cor()` function:

```
cor(IQ, WT, use = "pairwise.complete.obs")
```

```
## [1] -0.2917158
```

or by using a scatter plot:

```
plot(IQ, WT)
```



and by a formal test:

```
cor.test(IQ, WT)
```

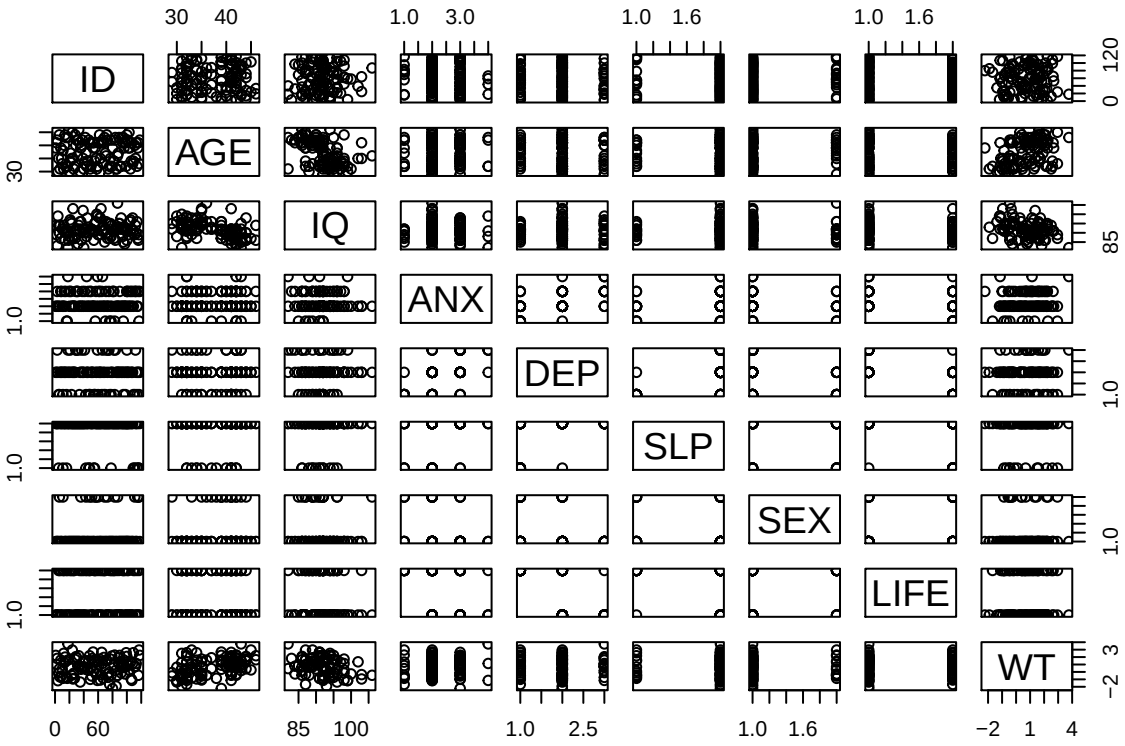
```
##
##  Pearson's product-moment correlation
##
## data:  IQ and WT
## t = -3.0192, df = 98, p-value = 0.003231
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.4616804 -0.1010899
## sample estimates:
##       cor
## -0.2917158
```

With some functions you can pass an entire data.frame rather than a list of variables:

```
cor(fem, use = "pairwise.complete.obs")
```

```
##               ID         AGE            IQ         ANX           DEP
## ID    1.00000000  0.03069077  0.0370598672 -0.02941825 -0.0554147209
## AGE   0.03069077  1.00000000 -0.4345435680  0.06734300 -0.0387049246
## IQ    0.03705987 -0.43454357  1.0000000000 -0.02323787 -0.0001307404
## ANX  -0.02941825  0.06734300 -0.0232378691  1.00000000  0.5437946347
## DEP  -0.05541472 -0.03870492 -0.0001307404  0.54379463  1.0000000000
## SLP  -0.07268743  0.02606547  0.0812993104  0.22317875  0.5248724551
## SEX   0.08999634  0.10609216 -0.0536558660 -0.21062493 -0.3058422258
## LIFE -0.05604349 -0.10300193 -0.0915396469 -0.34211268 -0.6139017253
## WT    0.02640131  0.41574411 -0.2917157832  0.11817532  0.0233742465
##               SLP         SEX        LIFE           WT
## ID   -0.072687434  0.08999634 -0.05604349  0.026401310
## AGE   0.026065468  0.10609216 -0.10300193  0.415744109
## IQ    0.081299310 -0.05365587 -0.09153965 -0.291715783
## ANX   0.223178752 -0.21062493 -0.34211268  0.118175321
## DEP   0.524872455 -0.30584223 -0.61390173  0.023374247
## SLP   1.000000000 -0.29053971 -0.35186578 -0.009259774
## SEX  -0.290539709  1.00000000  0.22316967 -0.027826514
## LIFE -0.351865775  0.22316967  1.00000000 -0.058605326
## WT   -0.009259774 -0.02782651 -0.05860533  1.000000000
```

```
pairs(fem)
```

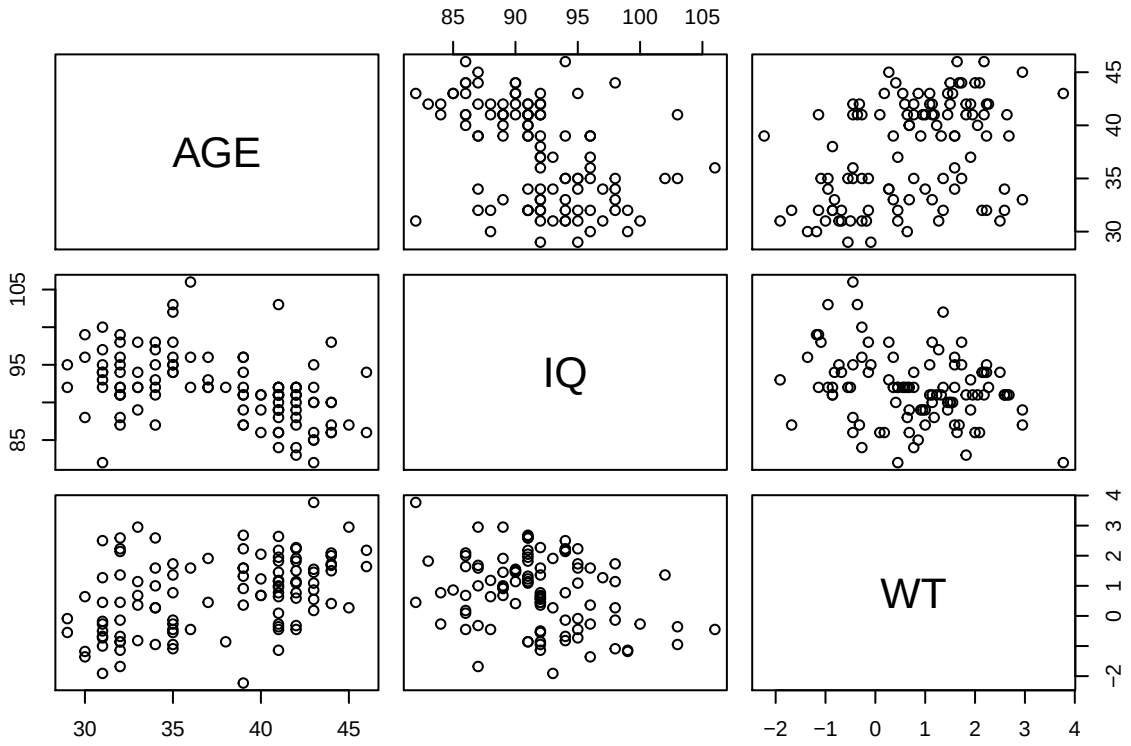


The output can be a little confusing particularly if it includes categorical or record identifying variables. To avoid this we can create a new object that contains only the columns we are interested in using the column binding `cbind()` function:

```
newfem <- cbind(AGE, IQ, WT)
cor(newfem, use = "pairwise.complete.obs")
```

```
##            AGE         IQ         WT
## AGE  1.0000000 -0.4345436  0.4157441
## IQ  -0.4345436  1.0000000 -0.2917158
## WT   0.4157441 -0.2917158  1.0000000
```

```
pairs(newfem)
```



When we have finished with the `newfem` object we can delete it:

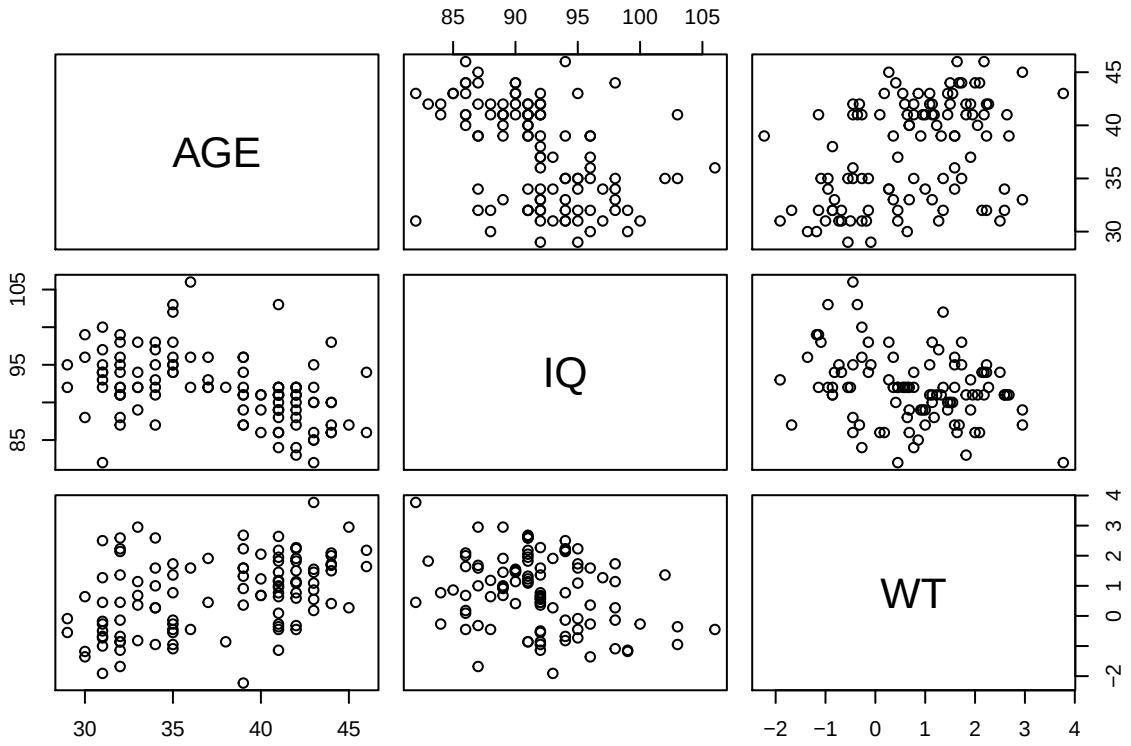
```
rm(newfem)
```

There was no real need to create the `newfem` object as we could have fed the output of the `cbind()` function directly to the `cor()` or `pairs()` function:

```
cor(cbind(AGE, IQ, WT), use = "pairwise.complete.obs")
```

```
##            AGE         IQ         WT
## AGE  1.0000000 -0.4345436  0.4157441
## IQ  -0.4345436  1.0000000 -0.2917158
## WT   0.4157441 -0.2917158  1.0000000
```

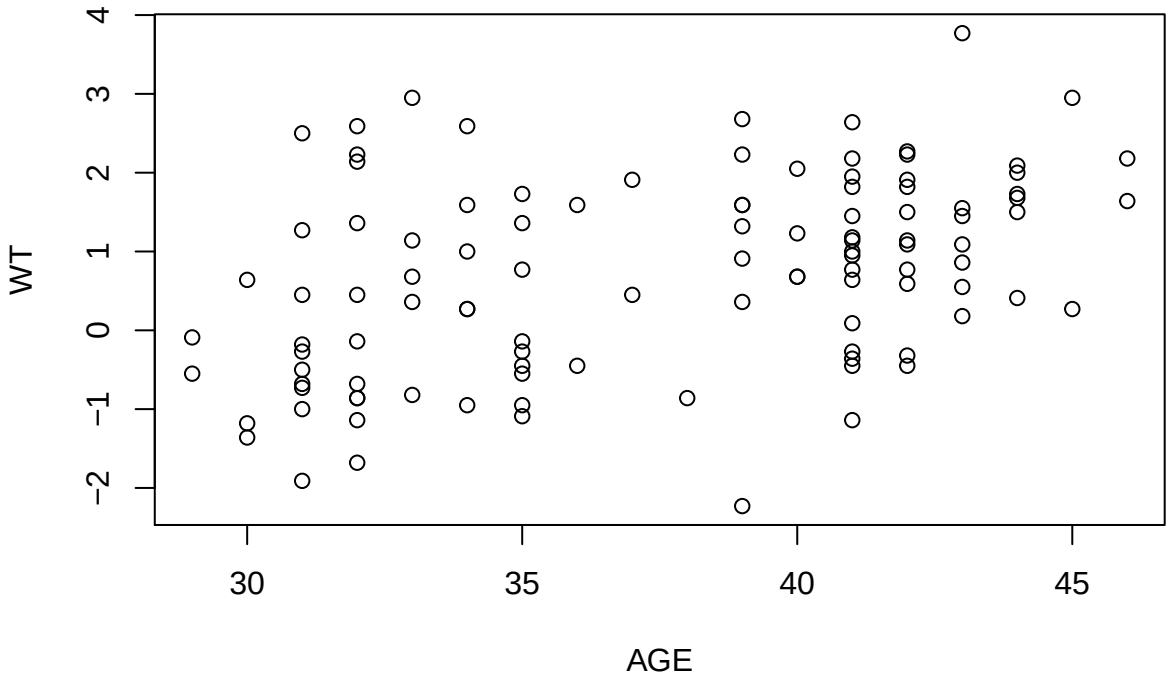
```
pairs(cbind(AGE, IQ, WT))
```



It is, however, easier to work with the `newfem` object rather than having to retype the `cbind()` function. This is particularly true if you wanted to continue with an analysis of just the three variables.

The relationship between `AGE` and `WT` can be plotted using the `plot()` function:

```
plot(AGE, WT)
```



And tested using the `cor()` and `cor.test()` functions:

```
cor(AGE, WT, use = "pairwise.complete.obs")
```

```
## [1] 0.4157441
```

```
cor.test(AGE, WT)
```

```
##
##  Pearson's product-moment correlation
##
## data:  AGE and WT
## t = 4.6841, df = 105, p-value = 8.457e-06
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.2452434 0.5612979
## sample estimates:
##       cor
## 0.4157441
```

Or by using the linear modelling `lm()` function:

```
summary(lm(WT ~ AGE))
```

```
##
## Call:
## lm(formula = WT ~ AGE)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.10678 -0.85922 -0.05453  0.71434  2.70874
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.25405    0.85547  -3.804  0.00024 ***
## AGE          0.10592    0.02261   4.684 8.46e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.128 on 105 degrees of freedom
##   (11 observations deleted due to missingness)
## Multiple R-squared:  0.1728, Adjusted R-squared:  0.165
## F-statistic: 21.94 on 1 and 105 DF,  p-value: 8.457e-06
```

We use the `summary()` function here to extract summary information from the output of the `lm()` function.

It is often more useful to use `lm()` to create an object:

```
fem.lm <- lm(WT ~ AGE)
```

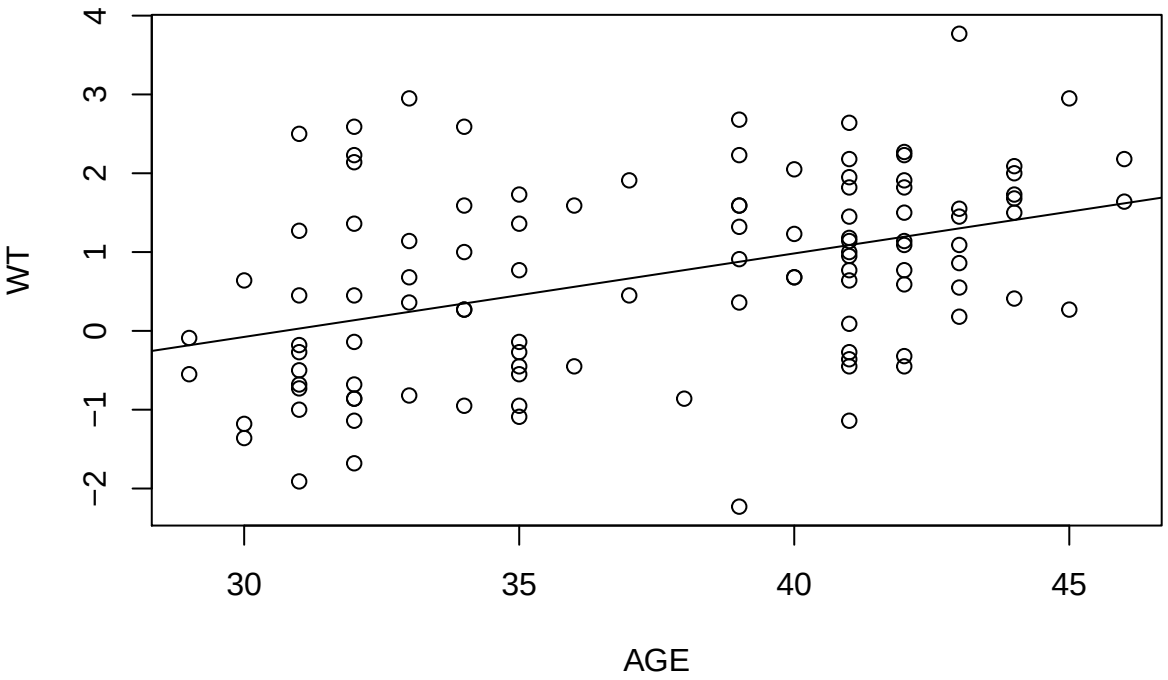
And use the output in other functions:

```
summary(fem.lm)
```

```
##
## Call:
## lm(formula = WT ~ AGE)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -3.10678 -0.85922 -0.05453  0.71434  2.70874
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.25405    0.85547  -3.804  0.00024 ***
## AGE          0.10592    0.02261   4.684 8.46e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.128 on 105 degrees of freedom
##   (11 observations deleted due to missingness)
## Multiple R-squared:  0.1728, Adjusted R-squared:  0.165
## F-statistic: 21.94 on 1 and 105 DF,  p-value: 8.457e-06
```

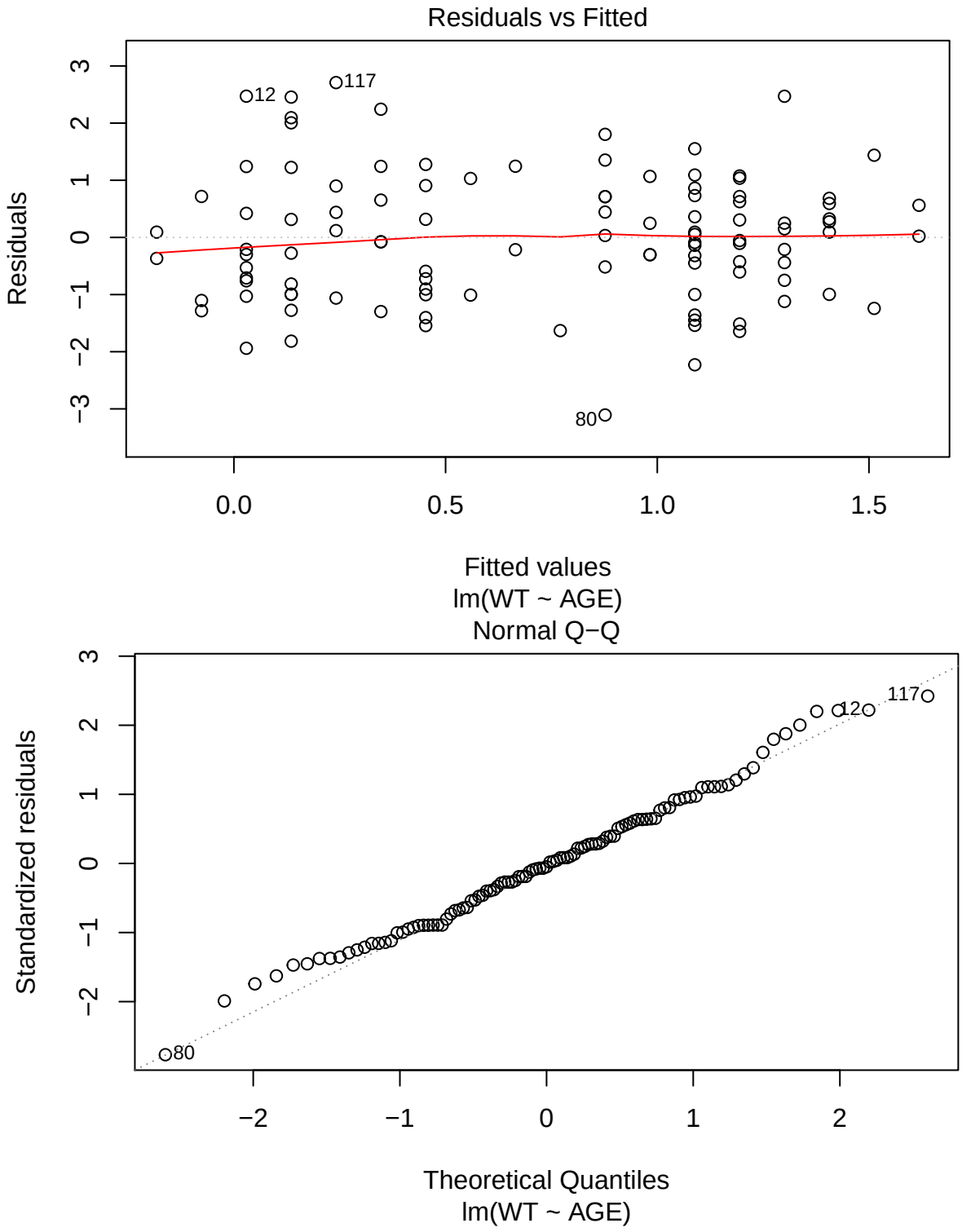
```
plot(AGE, WT)
abline(fem.lm)
```

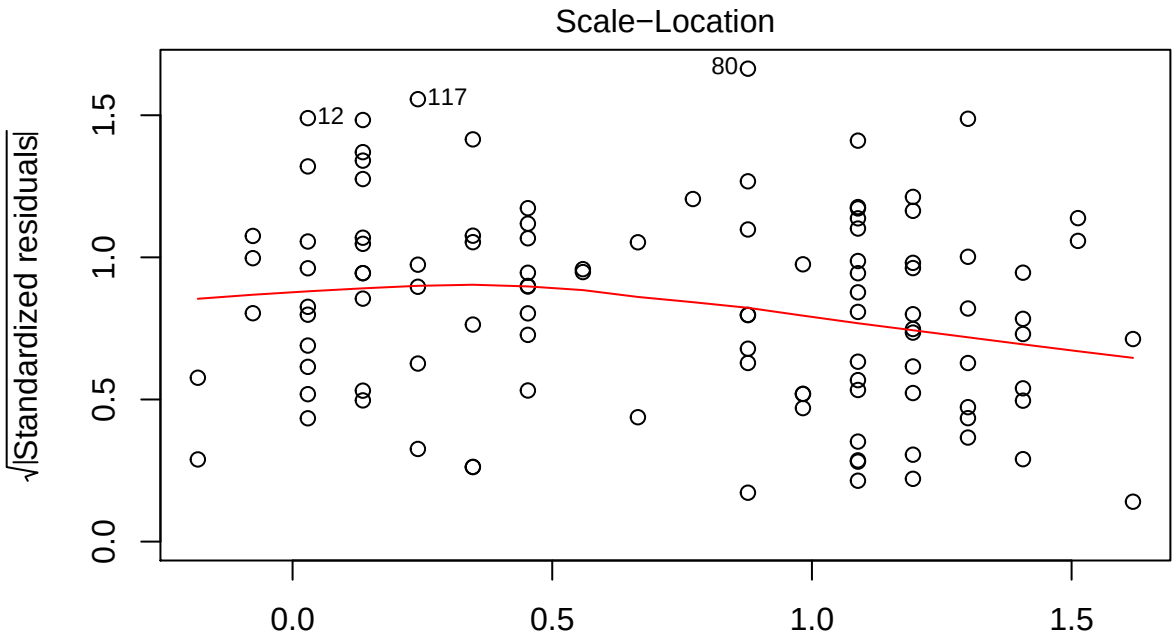


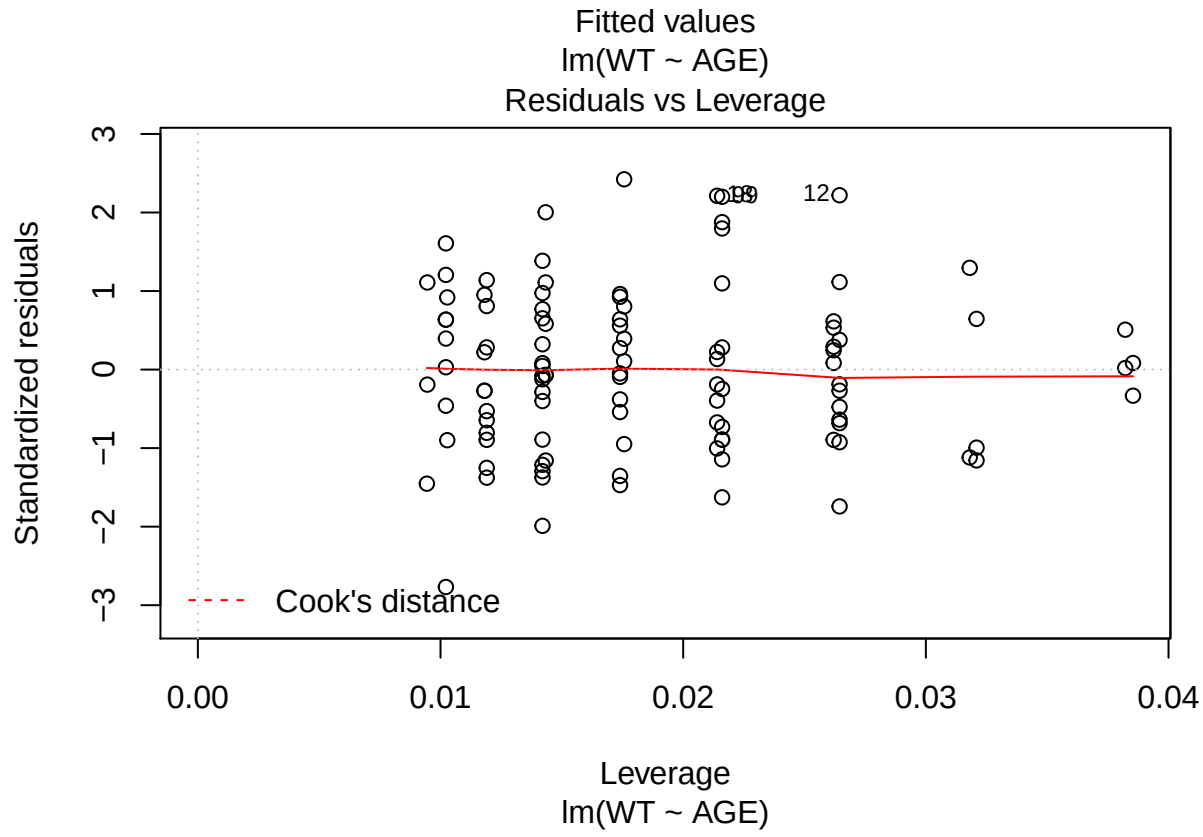
In this case we are passing the intercept and slope information held in the `fem.lm` object to the `abline()` function which draws a regression line. The `abline()` function adds to an existing plot. This means that you need to keep the scatter plot of `AGE` and `WT` open before issuing the `abline()` function call.

A useful function to apply to the `fem.lm` object is `plot()` which produces diagnostic plots of the linear model:

```
plot(fem.lm)
```


Residuals vs Fitted
Residuals
Fitted values
lm(WT ~ AGE)
12
117
80
Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(WT ~ AGE)



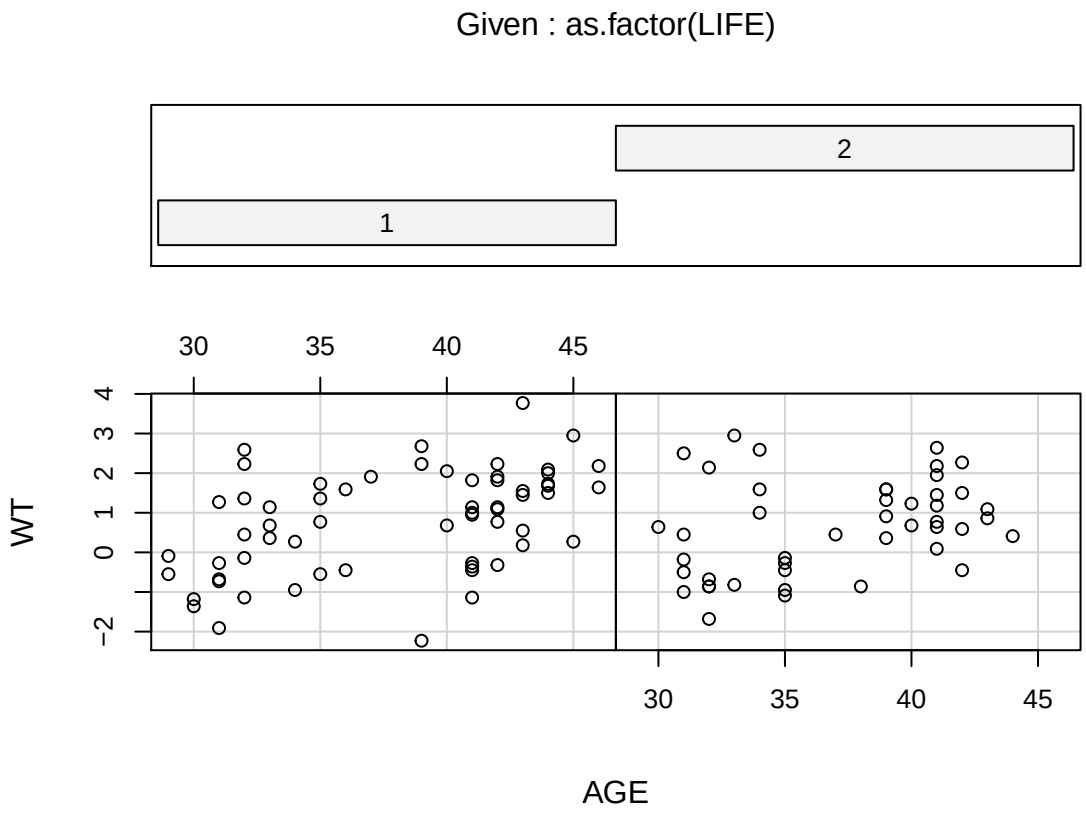


Objects created by the `lm()` function (or any of the modelling functions) can use up a lot of memory so we should remove them when we no longer need them:

```
rm(fem.lm)
```

It might be interesting to see whether a similar relationship exists between `AGE` and `WT` for those who have and have not considered suicide. This can be done using the `coplot()` function:

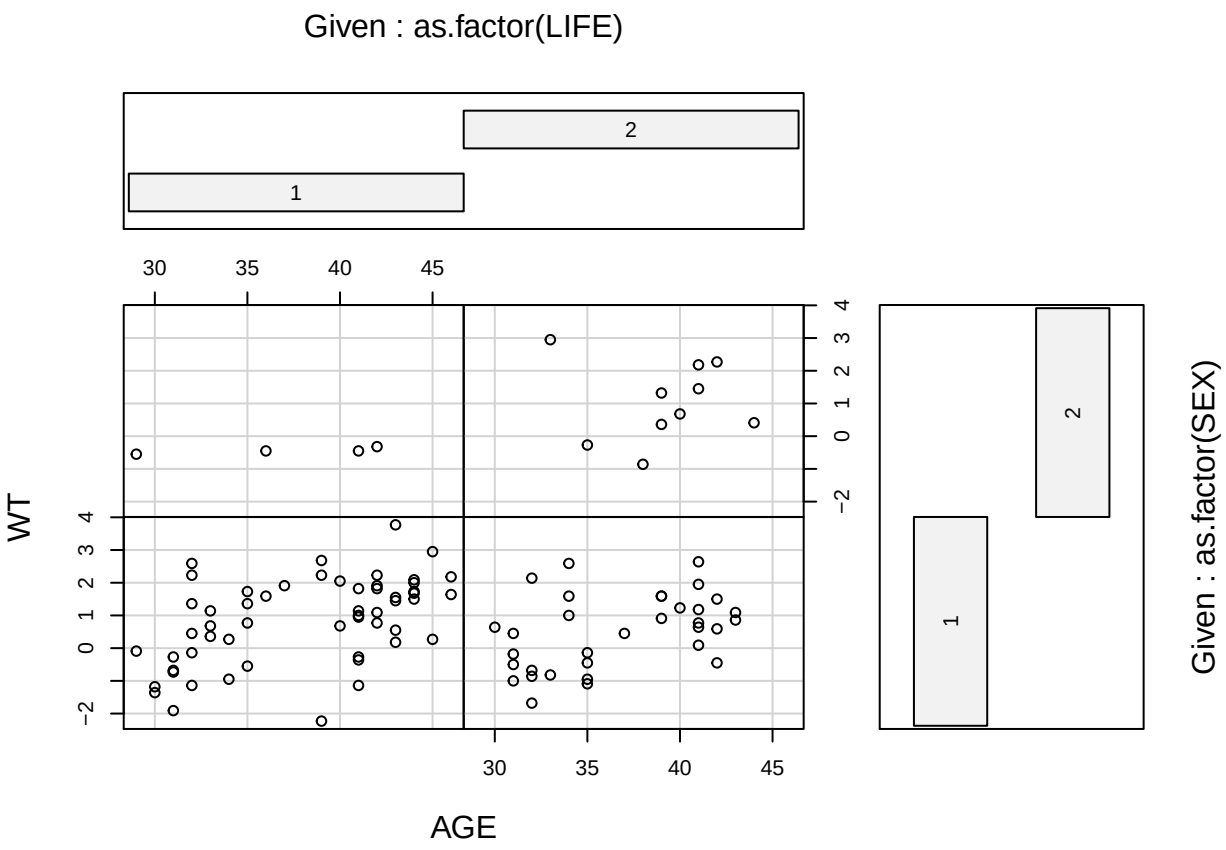
```
coplot(WT ~ AGE | as.factor(LIFE))
```




```
##
##  Missing rows: 21, 22, 31, 43, 44, 45, 69, 81, 101, 104, 114, 115
```

The two plots looks similar. We could also use `coplot()` to investigate the relationship between `AGE` and `WT` for categories of both `LIFE` and `SEX`:

```
coplot(WT ~ AGE | as.factor(LIFE) * as.factor(SEX))
```




```
##
##  Missing rows: 12, 17, 21, 22, 31, 43, 44, 45, 66, 69, 81, 101, 104, 105, 114, 115
```

although the numbers are too small for this to be useful here.

We used the `as.factor()` function with the `coplot()` function to ensure that `R` was aware that the `LIFE` and `SEX` columns hold categorical data.

We can check the way variables are stored using the `data.class()` function:

```
data.class(fem$SEX)
```

```
## [1] "numeric"
```

We can 'apply' this function to all columns in a data.frame using the `sapply()` function:

```
sapply(fem, data.class)
```

```
##        ID       AGE        IQ       ANX       DEP       SLP       SEX
## "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
##      LIFE        WT
## "numeric" "numeric"
```

The `sapply()` function is part of a group of functions that apply a specified function to data objects:

Function(s)	Applies a function to ...
`apply()`	rows and columns of matrices, arrays, and tables
`lapply()`	components of lists and data.frames
`sapply()`	components of lists and data.frames
`mapply()`	components of lists and data.frames
`tapply()`	subsets of data

Related functions are `aggregate()` which compute summary statistics for subsets of data, `by()` which applies a function to a data.frame split by factors, and `sweep()` which applies a function to an array.

The parameters of most `R` functions have default values. These are usually the most used and most useful parameter values for each function. The `cor.test()` function, for example, calculates *Pearson's product moment correlation coefficient* by default. This is an appropriate measure for data from a bivariate normal distribution. The `DEP` and `ANX` variables contain ordered data. An appropriate measure of correlation between `DEP` and `ANX` is *Kendall's tau*. This can be obtained using:

```
cor.test(DEP, ANX, method = "kendall")
```

```
##
##  Kendall's rank correlation tau
##
## data:  DEP and ANX
## z = 5.5606, p-value = 2.689e-08
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
##       tau
## 0.4950723
```

Before we finish we should save the `fem` data.frame so that next time we want to use it we will not have to bother with recoding the missing values to the special `NA` value. This is done with the `write.table()` function:

```
write.table(fem, file = "newfem.dat", row.names = FALSE)
```

Everything in `R` is either a function or an object. Even the command to quit `R` is a function:

```
q()
```

When you call the `q()` function you will be asked if you want to save the workspace image. If you save the workspace image then all of the objects and functions currently available to you will be saved. These will then be automatically restored the next time you start `R` in the current working directory.

For this exercise there is no need to save the workspace image so click the `No` or `Don't Save` button (GUI) or enter `n` when prompted to save the workspace image (terminal).

2.1 Summary

- `R` is a functional system. Everything is done by calling functions.
- `R` provides a large set of functions for descriptive statistics, charting, and statistical inference.
- Functions can be chained together so that the output of one function is the input of another function.
- `R` is an object oriented system. We can use functions to create objects that can then be manipulated or passed to other functions for subsequent analysis.

Chapter 3

Manipulating objects and creating new functions

In this exercise we will explore how to manipulate `R` objects and how to write functions that can manipulate and extract data and information from `R` objects and produce useful analyses.

Before we go any further we should start `R` and retrieve a dataset:

```
salex <- read.table("salex.dat", header = TRUE, na.strings = "9")
```

Missing values are coded as 9 throughout this dataset so we can use the `na.strings` parameter of the `read.table()` function to replace all 9's with the special `NA` code when we retrieve the dataset. Check that this works by examining the `salex` data.frame:

```
salex
```

```
##    ILL HAM BEEF EGGS MUSHROOM PEPPER PORKPIE PASTA RICE LETTUCE TOMATO
## 1    1   1    1    1        1      1       2     2    2       2      2
## 2    1   1    1    1        2      2       1     2    2       2      1
## 3    1   1    1    1        1      1       1     1    1       1      2
## 4    1   1    1    1        2      2       2     2    2       1      1
## 5    1   1    1    1        1      1       1     1    1       1      1
## 6    1   1    1    1        2      2       2     2    2       2      1
## 7    1   1    1    1        1      1       1     2    2       2      2
## 8    1   1    2    1        1      1       2     1    1       1      2
## 9    1   1    1    1        2      1       1     2    1       2      2
## 10   1   1    1    1        2      1       1     1    1       1      1
## 11   1   2    2    1        1      1       2     2    2       1      1
## 12   1   1    1    1        2      2       2     2    2       2      2
## 13   2   2    1    2        2      2       1     2    2       2      1
## 14   1   1    1    1        2      2       2     1    1       2      1
## 15   1   1    1    1        1      1       2     1    1       2      2
## 16   1   1    1    1        1      1       1     2    2       2      2
## 17   1   1    1    1        1      1       1     1    1       1      1
## 18   2   1    1    2        2      2       2     2    2       2      2
## 19   2   1    1    1        1      2       2     1    1       2      1
## 20   2   1    1    2        2      2       2     2    2       2      2
## 21   2   2    2    2        2      2       2     2    2       2      2
## 22   1   1    1    1        2      2       2     2    2       1      1
## 23   1   2    1    2        2      2       2     1    1       2      1
```

```
## 24   1   1   1   1    2    1    2    1    1    2    2
## 25   1   1   1   2    1    1    1    1    1    1    1
## 26   1   1   2   1    1    1    2    2    2    1    1
## 27   1   1   1   1    2    2    1    2    1    1    1
## 28   1   1   1   1    1    1    2    1    1    2    2
## 29   1   2   1   1    1   NA    2    1    1    1    1
## 30   1   1   1   2    2    2    1    2    2    2    2
## 31   1   1   1   1    1    2    2    1    1    2    2
## 32   1   1   1   1    1    2   NA    2    1    1    1
## 33   1   1   1   1    2    2    2    1    2    2    2
## 34   1   1   1   1    1    2    2    2    2    1    1
## 35   1   1   1   1    1    1    1    1    2    2    1
## 36   2   2   1   2    2    2    2    2    2    2    2
## 37   1   1   1   1    1    1    2    1    1    1    1
## 38   1   1   1   2    2    2    1    1    1    1    2
## 39   1   1   1   1    1    1    1    2    2    1    2
## 40   1   1   1   1    1    1    1    2    2    1    1
## 41   1   1   1   2    2    1    2    1    1    1    1
## 42   1   1   1   2    2    2    2    2    2    2    2
## 43   1   1   1   1    1    1    2    1    1    1    1
## 44   1   2   1   2    2    2    1    2    2    1    2
## 45   1   1   1   1    1    2    2    2    1    1    1
## 46   1   1   1   2    2    2    2    1    1    1    1
## 47   1   1   1   1    2    2    2    2    1    1    2
## 48   1   1   1   1    1   NA    1    1    1    2    2
## 49   1   1   1   1    2    1    2    2    1    1    1
## 50   1   2   1   1    2    2    2    1    2    2    1
## 51   2   2   1   2    2    2    2    2    2    2    2
## 52   2   1   1   2    2    2    2    1    2    2    1
## 53   2   1   1   2    2    2    1    2    2    2    1
## 54   2   1   1   2    1    2    1    2    2    2    1
## 55   2   1   1   1    1    1    2    2    1    2    2
## 56   2   1   1   2    2    2    2    2    2    2    1
## 57   2   1   1   1    1    1    1    2    2    2    2
## 58   2   1   1   1    2    2    1    2    1    2    2
## 59   2   1   1   2    2    2    2    2    2    2    2
## 60   2   2   2   2    2    2    1    2    2    2    2
## 61   2   1   1   2    2    2    1    2    2    2    2
## 62   2   1   2   2    2    2    2    2    2    1    1
## 63   1   1   1   1    1    1    2    2    2    2    1
## 64   2   1   1   2    2    2    2    2    2    2    2
## 65   2   1   1   1    1    2    1    2    1    2    2
## 66   2   2   1   2    2    2    2    2    2    2    2
## 67   2   2   1   2    2    2    2    2    2    2    2
## 68   2   1   1   2    1    1    1    1    2    2    1
## 69   2   2   1   2    2    2    2    2    2    2    2
## 70   2   2   1   2    2    2    2    2    2    2    2
## 71   1   1   2   2    2    2    1    2    1    2    2
## 72   2   1   2   1   NA   NA    2    2    2    2    1
## 73   1   1   1   1    2    2    1    2    2    2    2
## 74   1   1   2   1   NA   NA    2    1    1    1    1
## 75   1   1   2   2    2    1    2    1    2    1    1
## 76   1   1   1   1    2    2    1    1    2    2    2
## 77   1   1   1  NA   NA   NA    1    2    1    1    1
```

```
##    COLESLAW CRISPS PEACHCAKE CHOCOLATE FRUIT TRIFLE ALMONDS
## 1         2      2         2         2     2      2       2
## 2         2      2         2         2     2      2       2
## 3         2      1         2         1     2      2       2
## 4         2      2         2         1     2      2       2
## 5         1      2         2         1     2      1       2
## 6         1      1         2         1     2      2       2
## 7         1      1         1         2     2      2       2
## 8         1      1         2         2     2      1       2
## 9         2      2         2         2     2      1       2
## 10        1      1         2         2     2      1       1
## 11        2      2         2         2     2      2      NA
## 12        2      1         2         1     2      2       2
## 13        2      1         2         2     1      2      NA
## 14        1      1         2         2     2      1       2
## 15        1      1         2         2     2      1       1
## 16        1      2         2         2     2      2       2
## 17        1      2         2         2     2      2       2
## 18        2      2         2         2     2      2       2
## 19        1      1         2         2     1      2       2
## 20        2      2         2         1     2      2       2
## 21        2      2         2         2     2      2       2
## 22        2      1         2         1     2      2       2
## 23        1      2         2         2     2      2      NA
## 24        1      1         2         2     2      1       2
## 25        1      2         2         2     2      1      NA
## 26        1      2         2         2     2      1       2
## 27        1      1         1         1     2      1       2
## 28        2      1         2         2     2      2      NA
## 29        1      1         2         2     2      2      NA
## 30        2      2         2         2     2      2       2
## 31        2      2         2         2     2      2       2
## 32        2      2         2         2     2      2       2
## 33        1      2         2         2     2      2       2
## 34        1      2         2         2     2      1       2
## 35        1      2         2         2     2      1       2
## 36        2      2         2         2     2      2      NA
## 37        1      1         2         1     2      1       2
## 38        2      2         2         2     2      2       2
## 39        2      2         2         1     2      2       2
## 40        1      2         2         2     2      2       2
## 41        1      1         2         2    NA      1      NA
## 42        2      2         2         2     2      2      NA
## 43        1      1         2         2     2      2      NA
## 44        2      2         2         2     2      2       2
## 45        1      2         2         2     2      1       2
## 46        1      2         2         2     2      1       2
## 47        2      2         2        NA     2      1       2
## 48        2      1         2         2     2      2       2
## 49        1      1         2         2     2      1       2
## 50       NA      2         2         1     2      1       1
## 51        2      2         2         2     2      2      NA
## 52        2      2         2         1     2      2       1
## 53        2      2         2         2     1      2       2
```

```
## 54       2      2       2       2      2      2       2
## 55       2      1       2       2      2      2       2
## 56       2      2       2       2      1      2       2
## 57       1      2       2       2      2      2       1
## 58       2      1       1       2      2      2       2
## 59       2      1       1       2      2      1       2
## 60       2      2       2       2      2      2       2
## 61       2      1       2       2      2      1       1
## 62       2      1       2       2      2      2       2
## 63       1      2       2       1      1      2       2
## 64       2      2       2       2      2      2       2
## 65       1      1       2       2      2      1       2
## 66       2      2       2       2      2      1      NA
## 67       2      1       2       2      2      2       2
## 68       2      2       2       2      2      2       2
## 69       2      1       2       2      2      2       2
## 70       2      2       2       2      2      2       2
## 71       2      2       2       2      2      2       2
## 72       2      2       2       2      2      1       2
## 73       2      2       2       2      2      2       2
## 74       1      1       2       1      2      2       2
## 75       1      1       2       2      2      2      NA
## 76       2      2       2       2      2      2      NA
## 77       1      1       2       2      2      2       2
```

```
names(salex)
```

```
##  [1] "ILL"       "HAM"       "BEEF"      "EGGS"      "MUSHROOM"
##  [6] "PEPPER"    "PORKPIE"   "PASTA"     "RICE"      "LETTUCE"
## [11] "TOMATO"    "COLESLAW"  "CRISPS"    "PEACHCAKE" "CHOCOLATE"
## [16] "FRUIT"     "TRIFLE"    "ALMONDS"
```

This data comes from a food-borne outbreak. On Saturday 17th October 1992, eighty-two people attended a buffet meal at a sports club. Within fourteen to twenty-four hours, fifty-one of the participants developed diarrhoea, with nausea, vomiting, abdominal pain and fever.

The columns in the dataset are as follows:

ILL	Ill or not-ill
HAM	Baked ham
BEEF	Roast beef
EGGS	Eggs
MUSHROOM	Mushroom flan
PEPPER	Pepper flan
PORKPIE	Pork pie
PASTA	Pasta salad
RICE	Rice salad
LETTUCE	Lettuce
TOMATO	Tomato salad
COLESLAW	Coleslaw
CRISPS	Crisps
PEACHCAKE	Peach cake
CHOCOLATE	Chocolate cake
FRUIT	Tropical fruit salad
TRIFLE	Trifle
ALMONDS	Almonds

Data is available for seventy-seven of the eighty-two people who attended the sports club buffet. All of the variables are coded 1=yes, 2=no.

We can use the `attach()` function to make it easier to access our data:

```
attach(salex)
```

```
## The following objects are masked from salex (pos = 4):
##
##     ALMONDS, BEEF, CHOCOLATE, COLESLAW, CRISPS, EGGS, FRUIT, HAM,
##     ILL, LETTUCE, MUSHROOM, PASTA, PEACHCAKE, PEPPER, PORKPIE,
##     RICE, TOMATO, TRIFLE
```

The two-by-two table is a basic epidemiological tool. In analysing data from a food-borne outbreak collected as a retrospective cohort study, for example, we would tabulate each exposure (suspect foodstuffs) against the outcome (illness) and calculate risk ratios and confidence intervals. `R` has no explicit function to calculate risk ratios from two-by-two tables but we can easily write one ourselves.

The first step in writing such a function would be to create the two-by-two table. This can be done with the `table()` function. We will use a table of `HAM` by `ILL` as an illustration:

```
table(HAM, ILL)
```

This command produces the following output:

```
##    ILL
## HAM  1  2
##   1 46 17
##   2  5  9
```

We can manipulate the output directly but it is easier if we instruct `R` to save the output of the `table()` function in an object:

```
tab <- table(HAM, ILL)
```

The `tab` object contains the output of the `table()` function:

```
tab
```

```
##    ILL
## HAM  1  2
##   1 46 17
##   2  5  9
```

As it is stored in an object we can examine its contents on an item by item basis.

The `tab` object is an object of class `table`:

```
class(tab)
```

```
## [1] "table"
```

We can extract data from a table object by using indices or row and column co-ordinates:

```
tab[1,1]
```

```
## [1] 46
```

```
tab[1,2]
```

```
## [1] 17
```

```
tab[2,1]
```

```
## [1] 5
```

The numbers in the square brackets refer to the ***position*** (as row and column co-ordinates) of the data item in the table ***not*** the ***values*** of the variables. We can extract data using the values of the row and column variables by enclosing the index values in double quotes (“). For example:

```
tab["1","1"]
```

```
## [1] 46
```

The two methods of extracting data may be combined. For example:

```
tab[1,"1"]
```

```
## [1] 46
```

We can calculate a risk ratio using the extracted data:

```
(tab[1,1]/(tab[1,1]+tab[1,2]))/(tab[2,1]/(tab[2,1]+tab[2,2]))
```

Which returns a risk ratio of

```
## [1] 2.044444
```

This is a tedious calculation to have to type in every time you need to calculate a risk ratio from a two-by-two table. It would be better to have a function that calculates and displays the risk ratio automatically. Fortunately, `R` allows us to do just that.

The `function()` function allows us to create new functions in `R`:

```
tab2by2 <- function(exposure, outcome) {}
```

This creates an empty function called `tab2by2` that expects two parameters called `exposure` and `outcome`. We could type the whole function in at the `R` command prompt but it is easier to use a text editor:

```
fix(tab2by2)
```

This will start an editor with the empty `tab2by2()` function already loaded. We can now edit this function to make it do something useful:

```
function(exposure, outcome)
  {
  tab <- table(exposure, outcome)
  a <- tab[1,1]
  b <- tab[1,2]
  c <- tab[2,1]
  d <- tab[2,2]
  rr <- (a / (a + b)) / (c / (c + d))
  print(tab)
  print(rr)
  }
```

Once you have made the changes shown above, check your work, save the file, and quit the editor. Before proceeding we should examine the `tab2by2()` function to make sure we understand what the function will do:

- The first line defines `tab2by2` as a function that expects to be given two parameters which are called `exposure` and `outcome`.
- The body of the function (i.e. the work of the function) is enclosed within curly brackets (`{}`).
- The first line of the body of the function creates a table object (`tab`) using the variables specified when the `tab2by2()` function is called (these are the parameters `exposure` and `outcome`).

- The next line creates four new objects (called `a`, `b`, `c`, and `d`) which contain the values of the four cells in the two-by-two table.
- The following line calculates the risk ratio using the objects `a`, `b`, `c`, and `d` and stores the result of the calculation in an object called `rr`.
- The final two lines print the contents of the `tab` and `rr` objects.

Let's try the `tab2by2()` function with our test data:

```
tab2by2(HAM, ILL)
```

```
##          outcome
## exposure  1  2
##        1 46 17
##        2  5  9
## [1] 2.044444
```

The `tab2by2()` function displays a table of `HAM` by `ILL` followed by the risk ratio calculated from the data in the table.

Try producing another table:

```
tab2by2(PASTA, ILL)
```

```
##          outcome
## exposure  1  2
##        1 25  3
##        2 26 23
## [1] 1.682692
```

Have a look at the `R` objects available to you:

```
ls()
```

```
## [1] "fem"     "salex"   "tab"     "tab2by2"
```

Note that there are no `a`, `b`, `c`, `d`, or `rr` objects.

Examine the `tab` object:

```
tab
```

```
##    ILL
## HAM  1  2
##   1 46 17
##   2  5  9
```

This is the table of `HAM` by `ILL` that you created earlier ***not*** the table of `PASTA` by `ILL` that was created by the `tab2by2()` function.

The `tab`, `a`, `b`, `c`, `d`, and `rr` objects in the `tab2by2()` function are local to that function and do not change anything outside of that function. This means that the `tab` object inside the function is independent of any object of the same name outside of the function.

When a function completes its work, all of the objects that are local to that function are automatically removed. This is useful as it means that you can use object names inside functions that will not interfere with objects of the same name that are stored elsewhere. It also means that you do not clutter up the `R` workspace with temporary objects.

Just to prove that `tab` in the `tab2by2()` function exists only in the `tab2by2()` function we can delete the tab object from the `R` workspace:

```
rm(tab)
```

Now try another call to the `tab2by2()` function:

```
tab2by2(FRUIT, ILL)
```

```
##          outcome
## exposure  1  2
##        1  1  4
##        2 49 22
## [1] 0.2897959
```

Now list the `R` objects available to you:

```
ls()
```

```
## [1] "fem"     "salex"   "tab2by2"
```

Note that there are no `tab`, `a`, `b`, `c`, `d`, or `rr` objects.

The `tab2by2()` function is very limited. It only displays a table and calculates and displays a simple ratio. A more useful function would also calculate and display a confidence interval for the risk ratio. This is what we will do now. Use the `fix()` function to edit the `tab2by2()` function:

```
fix(tab2by2)
```

We can now edit this function to calculate and display a 95% confidence interval for the risk ratio.

```
function(exposure, outcome) {
  tab <- table(exposure, outcome)
  a <- tab[1,1]
  b <- tab[1,2]
  c <- tab[2,1]
  d <- tab[2,2]
  rr <- (a / (a + b)) / (c / (c + d))
  se.log.rr <- sqrt((b / a) / (a + b) + (d / c) / (c + d))
  lci.rr <- exp(log(rr) - 1.96 * se.log.rr)
  uci.rr <- exp(log(rr) + 1.96 * se.log.rr)
  print(tab)
  print(rr)
  print(lci.rr)
  print(uci.rr)
}
```

Once you have made the changes shown above, check your work, save the file, and quit the editor. We should test our revised function:

```
tab2by2(EGGS, ILL)
```

which produces the following output:

```
##          outcome
## exposure  1  2
##        1 40  6
##        2 10 20
## [1] 2.608696
## [1] 1.553564
## [1] 4.38044
```

The function works but the output could be improved. Use the `fix()` function to edit the `tab2by2()` function:

```
function(exposure, outcome) {
  tab <- table(exposure, outcome)
  a <- tab[1,1]
  b <- tab[1,2]
  c <- tab[2,1]
  d <- tab[2,2]
  rr <- (a / (a + b)) / (c / (c + d))
  se.log.rr <- sqrt((b / a) / (a + b) + (d / c) / (c + d))
  lci.rr <- exp(log(rr) - 1.96 * se.log.rr)
  uci.rr <- exp(log(rr) + 1.96 * se.log.rr)
  print(tab)
  cat("\nRR :", rr,
      "\n95% CI :", lci.rr, uci.rr, "\n")
}
```

Once you have made the changes shown above, save the file and quit the editor.

Now we can test our function again:

```
tab2by2(EGGS, ILL)
```

Which produces the following output:

```
##          outcome
## exposure  1  2
##        1 40  6
##        2 10 20
##
## RR : 2.608696
## 95% CI : 1.553564 4.38044
```

The `tab2by2()` function displays output but does not behave like a standard `R` function in the sense that you cannot save the results of the `tab2by2()` function into an object:

```
test2by2 <- tab2by2(EGGS, ILL)
```

```
##          outcome
## exposure  1  2
##        1 40  6
##        2 10 20
##
## RR : 2.608696
## 95% CI : 1.553564 4.38044
```

displays output but does not save anything in the `test2by2` object:

```
test2by2
```

```
## NULL
```

The returned value (`NULL`) means that `test2by2` is an empty object. We will not worry about this at the moment as the `tab2by2()` function is good-enough for our current purposes. In Exercise 6 we will explore how to make our own functions behave like standard `R` functions.

We will now add the calculation of the odds ratio and its 95% confidence interval to the `tab2by2()` function using the `fix()` function.

There are two ways of doing this. We could either calculate the odds ratio from the table and use (e.g.) the method of Woolf to calculate the confidence interval:

```
or <- (a / b) / (c / d)
se.log.or <- sqrt(1 / a + 1 / b + 1 / c + 1 / d)
lci.or <- exp(log(or) - 1.96 * se.log.or)
uci.or <- exp(log(or) + 1.96 * se.log.or)
cat("\nOR     :", or,
    "\n95% CI :", lci.or, uci.or, "\n")
```

or use the output of the `fisher.test()` function:

```
ft <- fisher.test(tab)
cat("\nOR     :", ft$estimate,
    "\n95% CI :", ft$conf.int, "\n")
```

Note that we can refer to components of a function's output using the same syntax as when we refer to columns in a data.frame (e.g. `ft$estimate` to examine the estimate of the odds ratio from the `fisher.test()` function stored in the object `ft`).

The names of elements in the output of a standard function such as `fisher.test()` can be found in the documentation or the help system. For example:

```
help(fisher.test)
```

Output elements are listed under the `Value` heading.

Revise the `tab2by2()` function to include the calculation of the odds ratio and the 95% confidence interval. The revised function will look something like this:

```
function(exposure, outcome) {
  tab <- table(exposure, outcome)
  a <- tab[1,1]
  b <- tab[1,2]
  c <- tab[2,1]
  d <- tab[2,2]
  rr <- (a / (a + b)) / (c / (c + d))
  se.log.rr <- sqrt((b / a) / (a + b) + (d / c) / (c + d))
  lci.rr <- exp(log(rr) - 1.96 * se.log.rr)
  uci.rr <- exp(log(rr) + 1.96 * se.log.rr)
  or <- (a / b) / (c / d)
  se.log.or <- sqrt(1 / a + 1 / b + 1 / c + 1 / d)
  lci.or <- exp(log(or) - 1.96 * se.log.or)
  uci.or <- exp(log(or) + 1.96 * se.log.or)
  ft <- fisher.test(tab)
  cat("\n")
  print(tab)

  cat("\nRelative Risk     :", rr,
      "\n95% CI             :", lci.rr, uci.rr, "\n")

  cat("\nSample Odds Ratio :", or,
      "\n95% CI             :", lci.or, uci.or, "\n")

  cat("\nMLE Odds Ratio    :", ft$estimate,
      "\n95% CI              :",  ft$conf.int, "\n\n")
}
```

Once you have made the changes shown above, check your work, save the file, and quit the editor.

Test the `tab2by2()` function when you have added the calculation of the odds ratio and its 95% confidence interval.

Now that we have a function that will calculate risk ratios and odds ratios with confidence intervals from a two- by-two table we can use it to analyse the `salex` data:

```
tab2by2(HAM, ILL)
```

```
##
##          outcome
## exposure  1  2
##        1 46 17
##        2  5  9
##
## Relative Risk     : 2.044444
## 95% CI            : 0.9964841 4.194501
##
## Sample Odds Ratio : 4.870588
## 95% CI            : 1.428423 16.60756
##
## MLE Odds Ratio    : 4.75649
## 95% CI             : 1.22777 20.82921
```

```
tab2by2(BEEF, ILL)
```

```
##
##          outcome
## exposure  1  2
##        1 45 22
##        2  6  4
##
## Relative Risk     : 1.119403
## 95% CI            : 0.6568821 1.907592
##
## Sample Odds Ratio : 1.363636
## 95% CI            : 0.3485746 5.334594
##
## MLE Odds Ratio    : 1.357903
## 95% CI             : 0.2547114 6.428414
```

```
tab2by2(EGGS, ILL)
```

```
##
##          outcome
## exposure  1  2
##        1 40  6
##        2 10 20
##
## Relative Risk     : 2.608696
## 95% CI            : 1.553564 4.38044
##
## Sample Odds Ratio : 13.33333
## 95% CI            : 4.240168 41.92706
##
## MLE Odds Ratio    : 12.74512
```

```
## 95% CI             : 3.762787 50.05419
```

```
tab2by2(MUSHROOM, ILL)
```

```
##
##          outcome
## exposure  1  2
##        1 24  6
##        2 25 19
##
## Relative Risk     : 1.408
## 95% CI            : 1.028944 1.926697
##
## Sample Odds Ratio : 3.04
## 95% CI            : 1.037274 8.909506
##
## MLE Odds Ratio    : 2.995207
## 95% CI             : 0.9421008 10.7953
```

```
tab2by2(PEPPER, ILL)
```

```
##
##          outcome
## exposure  1  2
##        1 24  3
##        2 23 22
##
## Relative Risk     : 1.73913
## 95% CI            : 1.26876 2.383882
##
## Sample Odds Ratio : 7.652174
## 95% CI            : 2.013718 29.07844
##
## MLE Odds Ratio    : 7.448216
## 95% CI             : 1.861728 44.12015
```

```
tab2by2(PORKPIE, ILL)
```

```
##
##          outcome
## exposure  1  2
##        1 21  9
##        2 29 17
##
## Relative Risk     : 1.110345
## 95% CI            : 0.8044752 1.532509
##
## Sample Odds Ratio : 1.367816
## 95% CI            : 0.5113158 3.659032
##
## MLE Odds Ratio    : 1.362228
## 95% CI             : 0.4636016 4.190667
```

```
tab2by2(PASTA, ILL)
```

```
##
##          outcome
```

```
## exposure  1  2
##        1 25  3
##        2 26 23
##
## Relative Risk     : 1.682692
## 95% CI            : 1.255392 2.255433
##
## Sample Odds Ratio : 7.371795
## 95% CI            : 1.964371 27.66451
##
## MLE Odds Ratio    : 7.195422
## 95% CI             : 1.829867 42.07488
```

```
tab2by2(RICE, ILL)
```

```
##
##         outcome
## exposure  1  2
##        1 28  4
##        2 23 22
##
## Relative Risk     : 1.711957
## 95% CI            : 1.250197 2.344268
##
## Sample Odds Ratio : 6.695652
## 95% CI            : 2.017327 22.22335
##
## MLE Odds Ratio    : 6.532868
## 95% CI             : 1.852297 29.84928
```

```
tab2by2(LETTUCE, ILL)
```

```
##
##         outcome
## exposure  1  2
##        1 28  1
##        2 23 25
##
## Relative Risk     : 2.014993
## 95% CI            : 1.488481 2.727744
##
## Sample Odds Ratio : 30.43478
## 95% CI            : 3.826938 242.041
##
## MLE Odds Ratio    : 29.32825
## 95% CI             : 4.161299 1284.306
```

```
tab2by2(TOMATO, ILL)
```

```
##
##         outcome
## exposure  1  2
##        1 29  9
##        2 22 17
##
## Relative Risk     : 1.352871
```

```
## 95% CI             : 0.974698 1.877771
##
## Sample Odds Ratio : 2.489899
## 95% CI             : 0.9347213 6.632562
##
## MLE Odds Ratio    : 2.459981
## 95% CI             : 0.8467562 7.558026
```

```
tab2by2(COLESLAW, ILL)
```

```
##
##         outcome
## exposure  1  2
##        1 29  3
##        2 21 23
##
## Relative Risk     : 1.89881
## 95% CI             : 1.366876 2.63775
##
## Sample Odds Ratio : 10.5873
## 95% CI             : 2.806364 39.9417
##
## MLE Odds Ratio    : 10.26269
## 95% CI             : 2.600771 60.35431
```

```
tab2by2(CRISPS, ILL)
```

```
##
##         outcome
## exposure  1  2
##        1 21 10
##        2 30 16
##
## Relative Risk     : 1.03871
## 95% CI             : 0.7529065 1.433004
##
## Sample Odds Ratio : 1.12
## 95% CI             : 0.4258139 2.945888
##
## MLE Odds Ratio    : 1.118358
## 95% CI             : 0.3858206 3.340535
```

```
tab2by2(PEACHCAKE, ILL)
```

```
##
##         outcome
## exposure  1  2
##        1  2  2
##        2 49 24
##
## Relative Risk     : 0.744898
## 95% CI             : 0.27594 2.010846
##
## Sample Odds Ratio : 0.4897959
## 95% CI             : 0.06497947 3.691936
##
```

```
## MLE Odds Ratio    : 0.4947099
## 95% CI             : 0.03393887 7.209143
```

```
tab2by2(CHOCOLATE, ILL)
```

```
##
##         outcome
## exposure  1  2
##        1 12  2
##        2 38 24
##
## Relative Risk     : 1.398496
## 95% CI            : 1.045064 1.871456
##
## Sample Odds Ratio : 3.789474
## 95% CI            : 0.7791326 18.43089
##
## MLE Odds Ratio    : 3.733535
## 95% CI             : 0.7318646 37.28268
```

```
tab2by2(FRUIT, ILL)
```

```
##
##         outcome
## exposure  1  2
##        1  1  4
##        2 49 22
##
## Relative Risk     : 0.2897959
## 95% CI            : 0.04985828 1.684408
##
## Sample Odds Ratio : 0.1122449
## 95% CI            : 0.01185022 1.06318
##
## MLE Odds Ratio    : 0.1157141
## 95% CI             : 0.002240848 1.256134
```

```
tab2by2(TRIFLE, ILL)
```

```
##
##         outcome
## exposure  1  2
##        1 19  5
##        2 32 21
##
## Relative Risk     : 1.311198
## 95% CI            : 0.9718621 1.769016
##
## Sample Odds Ratio : 2.49375
## 95% CI            : 0.8067804 7.708156
##
## MLE Odds Ratio    : 2.465794
## 95% CI             : 0.7363311 9.778463
```

```
tab2by2(ALMONDS, ILL)
```

```
##
```

```
##          outcome
## exposure  1  2
##        1  3  3
##        2 38 19
##
## Relative Risk     : 0.75
## 95% CI            : 0.3300089 1.7045
##
## Sample Odds Ratio : 0.5
## 95% CI            : 0.09203498 2.716358
##
## MLE Odds Ratio    : 0.505905
## 95% CI             : 0.06170211 4.141891
```

Make a note of any positive associations (i.e. with a risk ratio > 1 with a 95% confidence intervals that does not include one). We will use these for the next exercise when we will use logistic regression to analyse this data.

Save the `tab2by2()` function:

```
save(tab2by2, file = "tab2by2.r")
```

We can now quit `R`:

```
q()
```

For this exercise there is no need to save the workspace image so click the **No** or **Don't Save** button (GUI) or enter **n** when prompted to save the workspace image (terminal).

3.1 Summary

- `R` objects contain information that can be examined and manipulated.
- `R` can be extended by writing new functions.
- New functions can perform simple or complex data analysis.
- New functions can be composed of parts of existing function.
- New functions can be saved and used in subsequent `R` sessions.
- Objects defined within functions are local to that function and only exist while that function is being used. This means that you can re-use meaningful names within functions without them interfering with each other.

Chapter 4

Logistic regression and stratified analysis

In this exercise we will explore how R handles generalised linear models using the example of logistic regression. We will continue using the salex dataset. Start R and retrieve the salex dataset:

```
salex <- read.table("salex.dat", header = TRUE, na.strings = "9")
```

When we analysed this data using two-by-two tables and examining the risk ratio and 95% confidence interval associated with each exposure we found many significant positive associations:

Variable	RR	95% CI
EGGS	2.61	1.55, 4.38
MUSHROOM	1.41	1.03, 1.93
PEPPER	1.74	1.27, 2.38
PASTA	1.68	1.26, 2.26
RICE	1.72	1.25, 2.34
LETTUCE	2.01	1.49, 2.73
COLESLAW	1.89	1.37, 2.64
CHOCOLATE	1.39	1.05, 1.87

Some of these associations may be due to *confounding* in the data. We can use logistic regression to help us identify independent associations.

Logistic regression requires the dependent variable to be either 0 or 1. In order to perform a logistic regression we must first recode the ILL variable so that 0=no and 1=yes:

```
table(salex$ILL)
```

```
##
##  1  2
## 51 26
```

```
salex$ILL[salex$ILL == 2] <- 0
table(salex$ILL)
```

```
##
##  0  1
## 26 51
```

We could work with our data as it is but if we wanted to calculate odds ratios and confidence intervals we would calculate their reciprocals (i.e. odds ratios for non-exposure rather than for exposure). This is because of the way the data has been coded (1=yes, 2=no).

In order to calculate meaningful odds ratios the exposure variables should also be coded 0=no, 1=yes. The actual codes used are not important as long as the value used for 'exposed' is one greater than the value used for 'not exposed'.

We could issue a series of commands similar to the one we have just used to recode the `ILL` variable. This is both tedious and unnecessary as the structure of the dataset (i.e. all variables are coded identically) allows us to recode all variables with a single command:

```
salex <- read.table("salex.dat", header = TRUE, na.strings = "9")
salex[1:5, ]
```

```
##   ILL HAM BEEF EGGS MUSHROOM PEPPER PORKPIE PASTA RICE LETTUCE TOMATO
## 1   1   1    1    1        1      1       2     2    2       2      2
## 2   1   1    1    1        2      2       1     2    2       2      1
## 3   1   1    1    1        1      1       1     1    1       1      2
## 4   1   1    1    1        2      2       2     2    2       1      1
## 5   1   1    1    1        1      1       1     1    1       1      1
##   COLESLAW CRISPS PEACHCAKE CHOCOLATE FRUIT TRIFLE ALMONDS
## 1        2      2         2         2     2      2       2
## 2        2      2         2         2     2      2       2
## 3        2      1         2         1     2      2       2
## 4        2      2         2         1     2      2       2
## 5        1      2         2         1     2      1       2
```

```
salex <- 2 - salex
salex[1:5, ]
```

```
##   ILL HAM BEEF EGGS MUSHROOM PEPPER PORKPIE PASTA RICE LETTUCE TOMATO
## 1   1   1    1    1        1      1       0     0    0       0      0
## 2   1   1    1    1        0      0       1     0    0       0      1
## 3   1   1    1    1        1      1       1     1    1       1      0
## 4   1   1    1    1        0      0       0     0    0       1      1
## 5   1   1    1    1        1      1       1     1    1       1      1
##   COLESLAW CRISPS PEACHCAKE CHOCOLATE FRUIT TRIFLE ALMONDS
## 1        0      0         0         0     0      0       0
## 2        0      0         0         0     0      0       0
## 3        0      1         0         1     0      0       0
## 4        0      0         0         1     0      0       0
## 5        1      0         0         1     0      1       0
```

WARNING : The `attach()` function works with a copy of the data.frame rather than the original data.frame. Commands that manipulate variables in a data.frame may not work as expected if the data.frame has been attached using the `attach()` function.

It is better to manipulate data ***before*** attaching a data.frame. The `detach()` function may be used to remove an attachment prior to any data manipulation.

Many `R` users avoid using the `attach()` function altogether.

We can now use the generalised linear model `glm()` function to specify the logistic regression model:

```
salex.lreg <- glm(formula = ILL ~ EGGS + MUSHROOM + PEPPER + PASTA +
                  RICE + LETTUCE + COLESLAW + CHOCOLATE,
                  family = binomial(logit), data = salex)
```

The method used by the `glm()` function is defined by the `family` parameter. Here we specify `binomial` errors and a `logit` (logistic) linking function.

We have saved the output of the `glm()` function in the `salex.lreg` object. We can examine some basic information about the specified model using the `summary()` function:

```
summary(salex.lreg)
```

```
##
## Call:
## glm(formula = ILL ~ EGGS + MUSHROOM + PEPPER + PASTA + RICE +
##     LETTUCE + COLESLAW + CHOCOLATE, family = binomial(logit),
##     data = salex)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.92036  -0.49869   0.06877   0.40906   2.07182
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.021864   0.676606  -2.988  0.00281 **
## EGGS         3.579366   1.267870   2.823  0.00476 **
## MUSHROOM    -3.584345   1.728999  -2.073  0.03817 *
## PEPPER       2.348074   1.428177   1.644  0.10015
## PASTA        1.774818   1.162762   1.526  0.12692
## RICE         0.114180   1.193840   0.096  0.92381
## LETTUCE      3.401828   1.234060   2.757  0.00584 **
## COLESLAW     0.763857   1.024373   0.746  0.45586
## CHOCOLATE    0.009782   1.314683   0.007  0.99406
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 91.246  on 69  degrees of freedom
## Residual deviance: 41.260  on 61  degrees of freedom
##   (7 observations deleted due to missingness)
## AIC: 59.26
##
## Number of Fisher Scoring iterations: 7
```

We will use *backwards elimination* to remove non-significant variables from the logistic regression model. Remember that previous commands can be recalled and edited using the up and down arrow keys – they do not need to be typed out in full each time.

`CHOCOLATE` is the least significant variable in the model so we will remove this variable from the model. Storing the output of the `glm()` function is useful as it allows us to use the `update()` function to add, remove, or modify variables without having to describe the model in full:

```
salex.lreg <- update(salex.lreg, . ~ . - CHOCOLATE)
summary(salex.lreg)
```

```
##
## Call:
## glm(formula = ILL ~ EGGS + MUSHROOM + PEPPER + PASTA + RICE +
##     LETTUCE + COLESLAW, family = binomial(logit), data = salex)
##
```

```
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.92561  -0.49859   0.07555   0.38723   2.07200
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -2.0223     0.6623  -3.053  0.00226 **
## EGGS          3.5890     1.2188   2.945  0.00323 **
## MUSHROOM     -3.5992     1.6885  -2.132  0.03305 *
## PEPPER        2.3544     1.4275   1.649  0.09910 .
## PASTA         1.7770     1.1215   1.585  0.11308
## RICE          0.1170     1.1388   0.103  0.91819
## LETTUCE       3.4109     1.2316   2.770  0.00561 **
## COLESLAW      0.7630     1.0224   0.746  0.45547
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 92.122  on 70  degrees of freedom
## Residual deviance: 41.273  on 63  degrees of freedom
##   (6 observations deleted due to missingness)
## AIC: 57.273
##
## Number of Fisher Scoring iterations: 7
```

RICE is now the least significant variable in the model so we will remove this variable from the model:

```
salex.lreg <- update(salex.lreg, . ~ . - RICE)
summary(salex.lreg)
```

```
##
## Call:
## glm(formula = ILL ~ EGGS + MUSHROOM + PEPPER + PASTA + LETTUCE +
##     COLESLAW, family = binomial(logit), data = salex)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8877  -0.4999   0.0786   0.3897   2.0697
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -2.0169     0.6600  -3.056  0.00224 **
## EGGS          3.6142     1.1944   3.026  0.00248 **
## MUSHROOM     -3.5508     1.6134  -2.201  0.02774 *
## PEPPER        2.3002     1.3200   1.743  0.08141 .
## PASTA         1.8230     1.0280   1.773  0.07617 .
## LETTUCE       3.4199     1.2273   2.787  0.00533 **
## COLESLAW      0.7611     1.0203   0.746  0.45571
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 92.122  on 70  degrees of freedom
```

```
## Residual deviance: 41.283  on 64  degrees of freedom
##   (6 observations deleted due to missingness)
## AIC: 55.283
##
## Number of Fisher Scoring iterations: 6
```

COLESLAW is now the least significant variable in the model so we will remove this variable from the model:

```
salex.lreg <- update(salex.lreg, . ~ . - COLESLAW)
summary(salex.lreg)
```

```
##
## Call:
## glm(formula = ILL ~ EGGS + MUSHROOM + PEPPER + PASTA + LETTUCE,
##     family = binomial(logit), data = salex)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.98481  -0.50486   0.08871   0.36910   2.06065
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.9957     0.6545  -3.049  0.00230 **
## EGGS          3.8152     1.1640   3.278  0.00105 **
## MUSHROOM     -3.4008     1.5922  -2.136  0.03269 *
## PEPPER        2.3520     1.3269   1.773  0.07631 .
## PASTA         1.9706     0.9922   1.986  0.04701 *
## LETTUCE       3.4786     1.2246   2.841  0.00450 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 92.982  on 71  degrees of freedom
## Residual deviance: 41.895  on 66  degrees of freedom
##   (5 observations deleted due to missingness)
## AIC: 53.895
##
## Number of Fisher Scoring iterations: 6
```

PEPPER is now the least significant variable in the model so we will remove this variable from the model:

```
salex.lreg <- update(salex.lreg, . ~ . - PEPPER)
summary(salex.lreg)
```

```
##
## Call:
## glm(formula = ILL ~ EGGS + MUSHROOM + PASTA + LETTUCE, family = binomial(logit),
##     data = salex)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0920  -0.5360   0.1109   0.4876   2.0056
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
```

```
## (Intercept)  -1.8676     0.6128  -3.048 0.002306 **
## EGGS          3.7094     1.0682   3.473 0.000515 ***
## MUSHROOM     -1.6165     1.0829  -1.493 0.135524
## PASTA         1.8440     0.9193   2.006 0.044864 *
## LETTUCE       3.2458     1.1698   2.775 0.005527 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 94.659  on 73  degrees of freedom
## Residual deviance: 45.578  on 69  degrees of freedom
##   (3 observations deleted due to missingness)
## AIC: 55.578
##
## Number of Fisher Scoring iterations: 6
```

MUSHROOM is now the least significant variable in the model so we will remove this variable from the model:

```
salex.lreg <- update(salex.lreg, . ~ . - MUSHROOM)
summary(salex.lreg)
```

```
##
## Call:
## glm(formula = ILL ~ EGGS + PASTA + LETTUCE, family = binomial(logit),
##     data = salex)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2024  -0.5108   0.2038   0.4304   2.0501
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.9710     0.6146  -3.207  0.00134 **
## EGGS          2.6391     0.7334   3.599  0.00032 ***
## PASTA         1.6646     0.8376   1.987  0.04689 *
## LETTUCE       3.1956     1.1516   2.775  0.00552 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 97.648  on 75  degrees of freedom
## Residual deviance: 50.529  on 72  degrees of freedom
##   (1 observation deleted due to missingness)
## AIC: 58.529
##
## Number of Fisher Scoring iterations: 6
```

There are now no non-significant variables in the model.

Unfortunately R does not present information on the model coefficients in terms of odds ratios and confidence intervals but we can write a function to calculate them for us.

The first step in doing this is to realise that the `salex.lreg` object contains essential information about the fitted model. To calculate odds ratios and confidence intervals we need the regression coefficients and their standard errors. Both:

```
summary(salex.lreg)$coefficients
```

```
##              Estimate Std. Error   z value     Pr(>|z|)
## (Intercept) -1.970967  0.6145691 -3.207071 0.0013409398
## EGGS         2.639115  0.7333899  3.598515 0.0003200388
## PASTA        1.664581  0.8375970  1.987330 0.0468858898
## LETTUCE      3.195594  1.1516159  2.774879 0.0055222320
```

and:

```
coef(summary(salex.lreg))
```

```
##              Estimate Std. Error   z value     Pr(>|z|)
## (Intercept) -1.970967  0.6145691 -3.207071 0.0013409398
## EGGS         2.639115  0.7333899  3.598515 0.0003200388
## PASTA        1.664581  0.8375970  1.987330 0.0468858898
## LETTUCE      3.195594  1.1516159  2.774879 0.0055222320
```

extract the data that we require. The preferred method is to use the `coef()` function. This is because some fitted models may return coefficients in a more complicated manner than (e.g.) those created by the `glm()` function. The `coef()` function provides a standard way of extracting this data from all classes of fitted objects.

We can store the `coefficients` data in a separate object to make it easier to work with:

```
salex.lreg.coeffs <- coef(summary(salex.lreg))
salex.lreg.coeffs
```

```
##              Estimate Std. Error   z value     Pr(>|z|)
## (Intercept) -1.970967  0.6145691 -3.207071 0.0013409398
## EGGS         2.639115  0.7333899  3.598515 0.0003200388
## PASTA        1.664581  0.8375970  1.987330 0.0468858898
## LETTUCE      3.195594  1.1516159  2.774879 0.0055222320
```

We can extract information from this object by addressing each piece of information by its row and column position in the object. For example:

```
salex.lreg.coeffs[2,1]
```

```
## [1] 2.639115
```

Is the regression coefficient for `EGGS`, and:

```
salex.lreg.coeffs[3,2]
```

```
## [1] 0.837597
```

is the standard error of the regression coefficient for `PASTA`. Similarly:

```
salex.lreg.coeffs[ ,1]
```

```
## (Intercept)        EGGS       PASTA     LETTUCE
##   -1.970967    2.639115    1.664581    3.195594
```

Returns the regression coefficients for all of the variables in the model, and:

```
salex.lreg.coeffs[ ,2]
```

```
## (Intercept)        EGGS       PASTA     LETTUCE
##   0.6145691   0.7333899   0.8375970   1.1516159
```

Returns the standard errors of the regression coefficients.

The table below shows the indices that address each cell in the table of regression coefficients:

```
matrix(salex.lreg.coeffs, nrow = 4, ncol = 4)
```

```
##           [,1]      [,2]      [,3]         [,4]
## [1,] -1.970967 0.6145691 -3.207071 0.0013409398
## [2,]  2.639115 0.7333899  3.598515 0.0003200388
## [3,]  1.664581 0.8375970  1.987330 0.0468858898
## [4,]  3.195594 1.1516159  2.774879 0.0055222320
```

We can use this information to calculate odds ratio sand 95% confidence intervals:

```
or <- exp(salex.lreg.coeffs[ ,1])
lci <- exp(salex.lreg.coeffs[ ,1] - 1.96 * salex.lreg.coeffs[ ,2])
uci <- exp(salex.lreg.coeffs[ ,1] + 1.96 * salex.lreg.coeffs[ ,2])
```

and make a single object that contains all of the required information:

```
lreg.or <- cbind(or, lci, uci)
lreg.or
```

```
##                     or       lci         uci
## (Intercept)  0.1393221 0.0417723   0.4646777
## EGGS        14.0008053 3.3256684  58.9423019
## PASTA        5.2834608 1.0231552  27.2832114
## LETTUCE     24.4246856 2.5559581 233.4018193
```

Chapter 5

Analysing some data with R

Chapter 6

Extending R with packages

Chapter 7

Making your own objects behave like R objects

Chapter 8

Writing your own graphical functions

Chapter 9

More graphical functions

Chapter 10

Computer intensive methods

Chapter 11

What now?