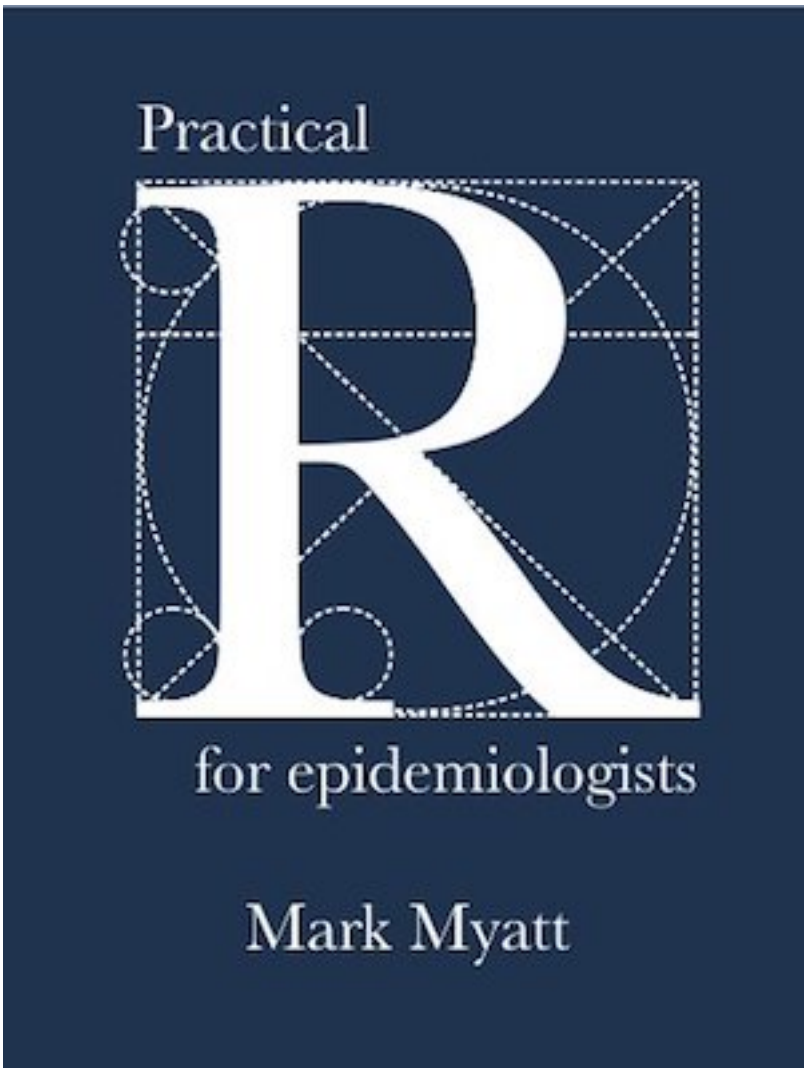
Practical R for Epidemiologists

Mark Myatt

2018-04-23

Contents

Welcome to Practical R for Epidemiologists



This is the website for *Practical R for Epidemiologists*. Visit the GitHub repository for this site or buy it as a Kindle ebook on Amazon.

Introduction

These notes are intended as a practical introduction to using the R environment for data analysis and graphics to work with epidemiological data. Topics covered include univariate statistics, simple statistical inference, charting data, two-by-two tables, stratified analysis, chi-square test for trend, logistic regression, survival analysis, computer-intensive methods, and extending R using user-provided functions. You should be able to follow the material if you are reasonably familiar with the mechanics of statistical estimation (e.g. calculation of odds ratios and confidence intervals) and require a system that can perform simple or complex analyses to your exact specifications.

These notes are split into ten sections:

Introduction: You are reading this section now!

Introducing R: Some information about the R system, the way the R system works, how to get a copy of R, and how to start R.

Exercise 1: Read a dataset, producing descriptive statistics, charts, and perform simple statistical inference. The aim of the exercise is for you to become familiar with R and some basic R functions and objects.

Exercise 2: In this exercise we 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.

Exercise 3: In this exercise we explore how R handles generalised linear models using the example of logistic regression as well as seeing how R can perform stratified (i.e. Mantel-Haenszel) analysis as well as analysing data arising from matched case-control studies.

Exercise 4: In this exercise we use R to analyse a small dataset using the methods introduced in the previous exercises.

Exercise 5: In this exercise we explore how R can be extended using add-in packages. Specifically, we will use an add-in package to perform a survival analysis.

Exercise 6: In this exercise we explore how to make your own R functions behave like R objects so that they return a data-structure that can be manipulated or interrogated by other R functions.

Exercise 7: In this exercise we explore how you can use R to produce custom graphical

functions.

Exercise 8: In this exercise we explore some more graphical functions and create custom graphical functions that produce two variable plots, pyramid charts, Pareto charts, charts with error bars, and simple mesh-maps.

Exercise 9: In this exercise we explore ways of implementing computer-intensive methods, such as the bootstrap and computer based simulation, using standard `R` functions.

If you are interested in a system that is flexible, can be tailored to produce exactly the analysis you want, provides modern analytical facilities, and have a basic understanding of the mechanics of hypothesis testing and estimation then you should consider following this material.

Chapter 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.

1.1 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 current working directory.

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("~/prfe/fem.dat", header = TRUE)
```

will retrieve the data stored in the file named `fem.dat` stored in the `prfe` 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:\\prfe\\fem.dat", header = TRUE)
```

will retrieve the data that is stored in the file named `fem.dat` which is stored in the `prfe` 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:/prfe/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://~/prfe/fem.dat", header = TRUE)
```

will retrieve the data stored in the file named `fem.dat` stored in the `prfe` directory under the users home directory on UNIX-based systems.

All of the data files used in this section are stored in the `/prfe` directory Brixton Health's website. This means, for example, that you can use the `read.table()` function specifying

"http://www.brixtonhealth.com/prfe/fem.dat"

as the `URL` to retrieve the data that is stored in the file named `fem.dat` which is stored in the `/prfe` 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

## The following objects are masked from fem (pos = 5):
##
##     AGE, ANX, DEP, ID, IQ, LIFE, SEX, SLP, WT

## The following objects are masked from fem (pos = 10):
##
##     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`:

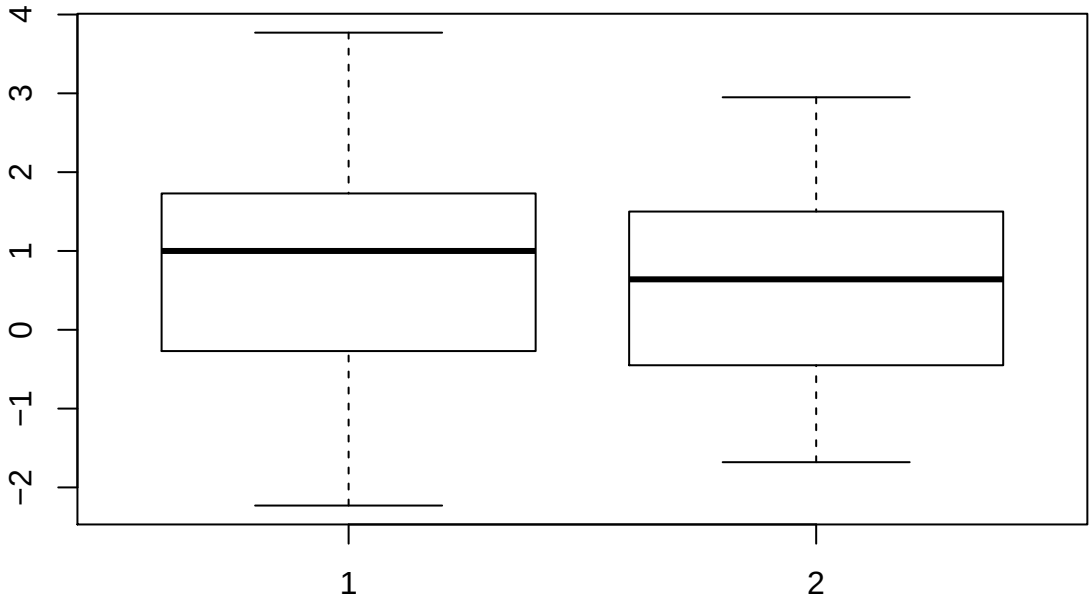
```
by(WT, LIFE, summary)
```

```
## 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
```

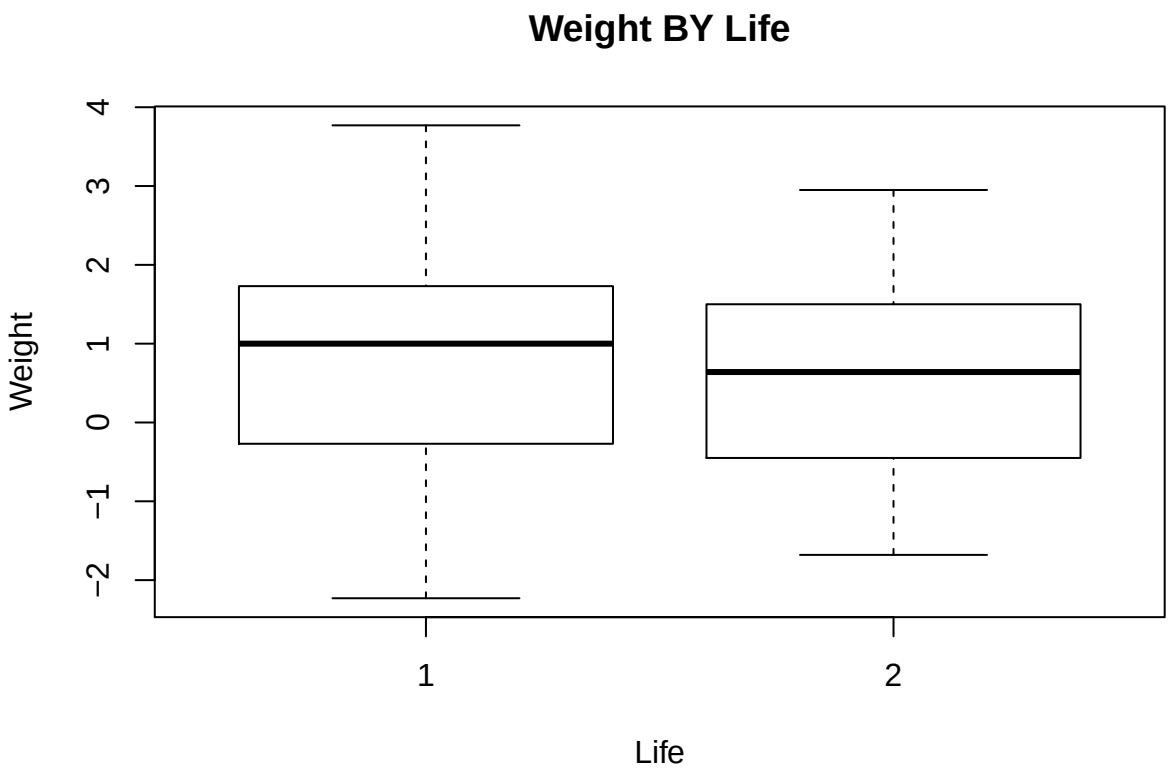
We can view the same data as a box and whisker plot:

```
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")
```

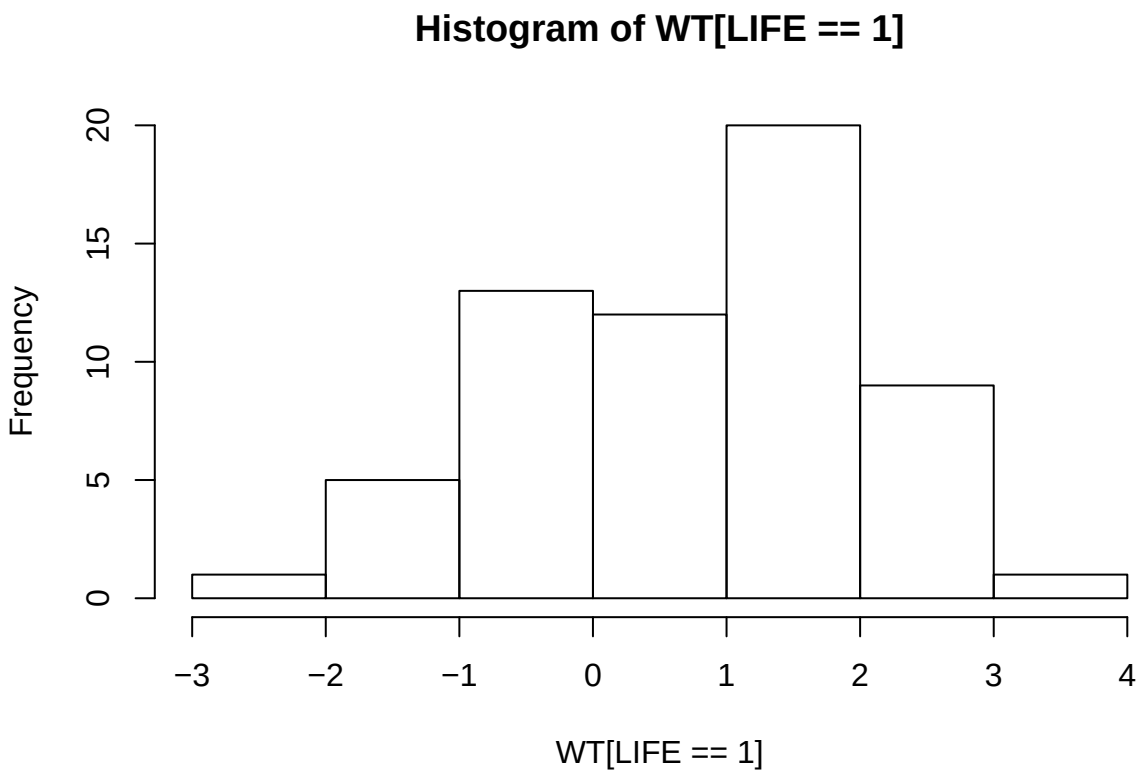


A more descriptive title might be “Weight Change BY Considered Suicide”.

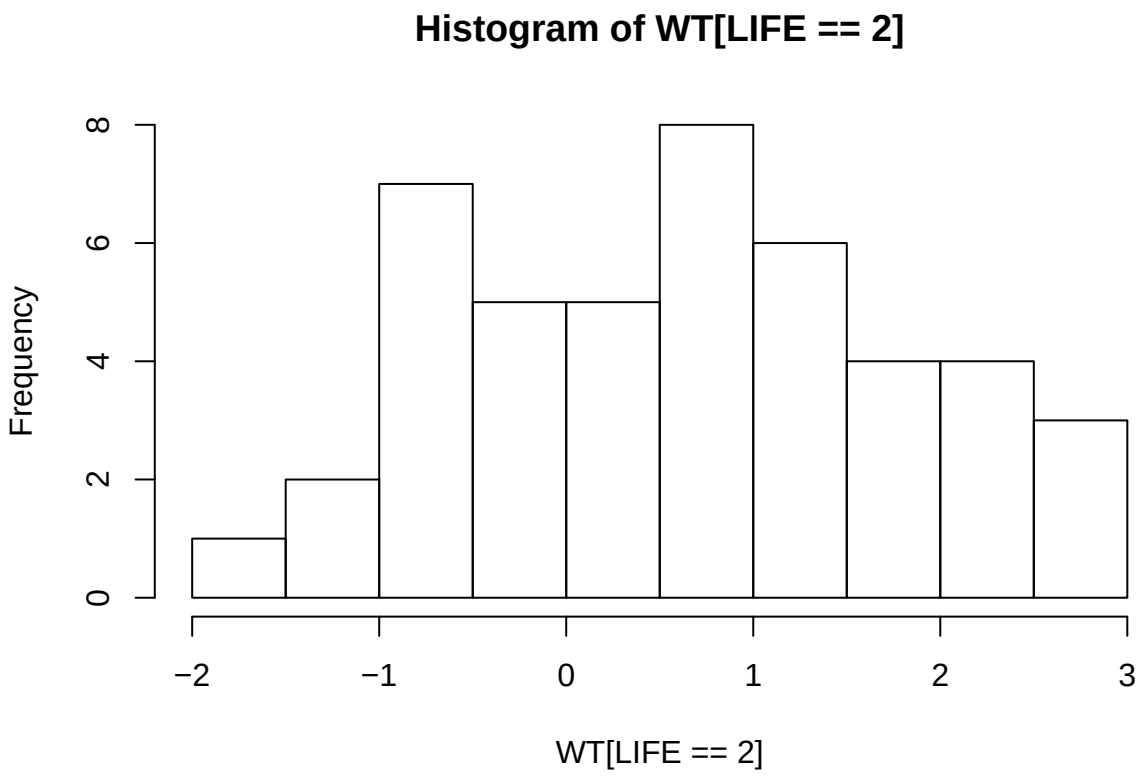
The groups do not seem to differ much in their medians and the distributions appear to be reasonably symmetrical about their medians with a similar spread of values.

We can look at the distribution as histograms:

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

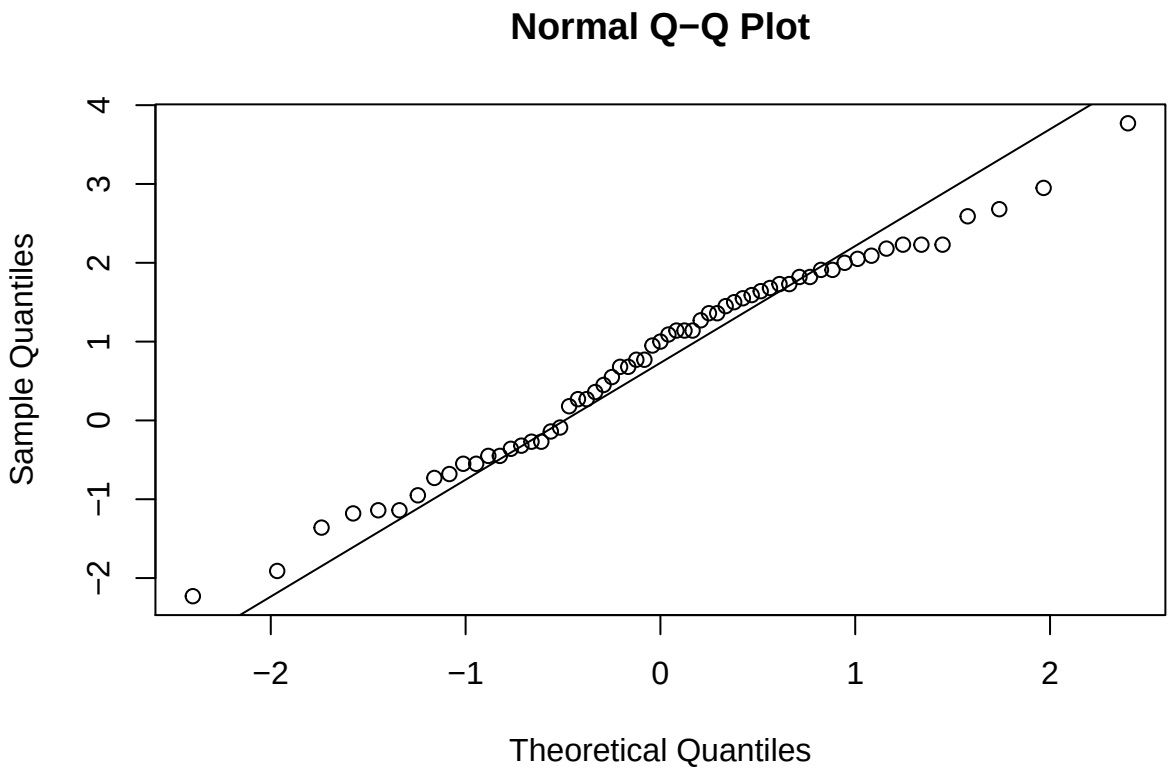



```
hist(WT[LIFE == 2])
```

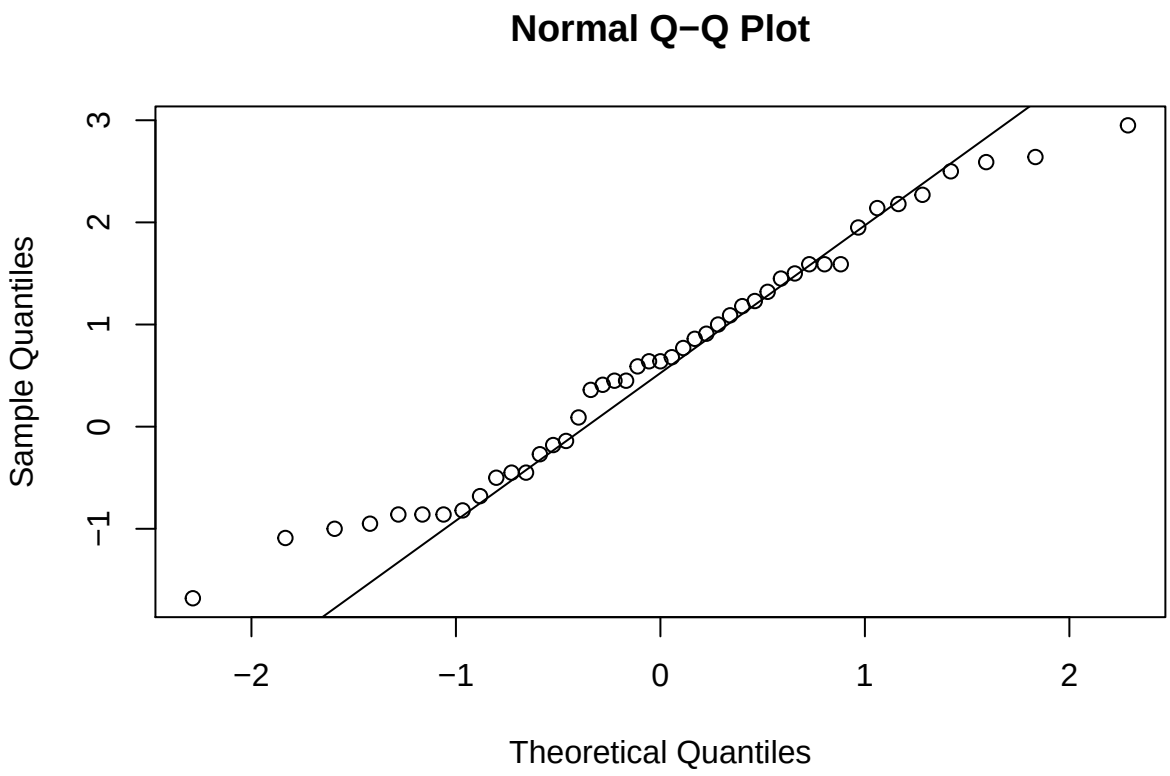


and check the assumption of normality using quantile-quantile plots:

```
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.

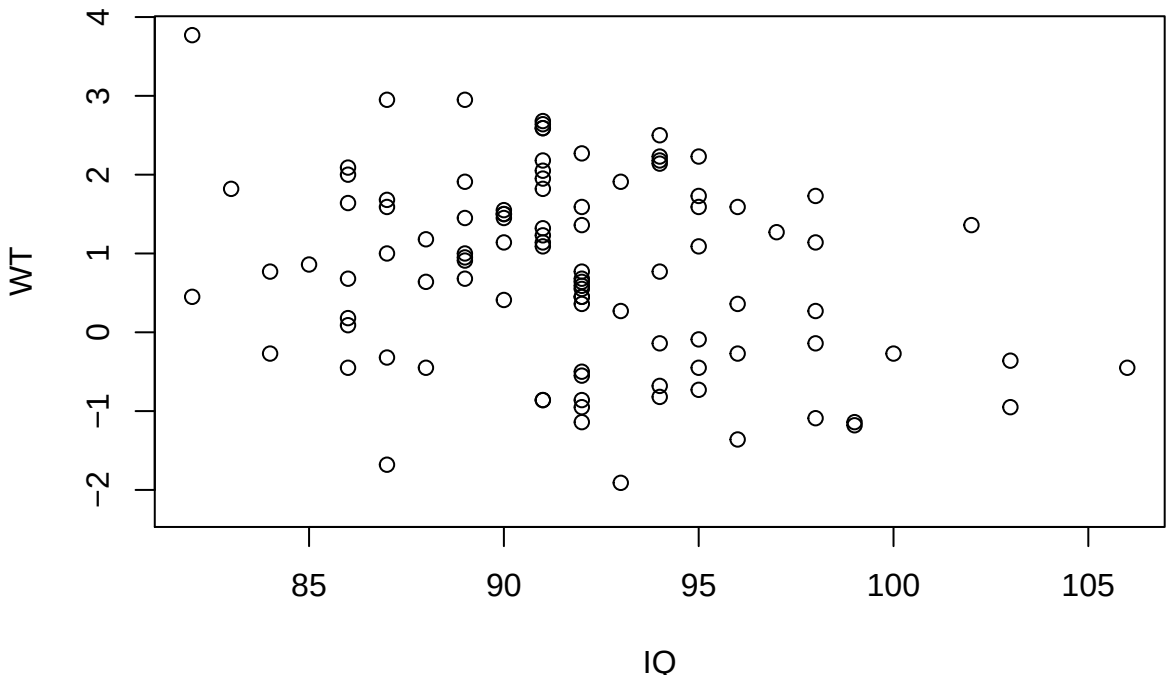
We can explore the correlation between two variables using the `cor()` function:

```
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
##
```

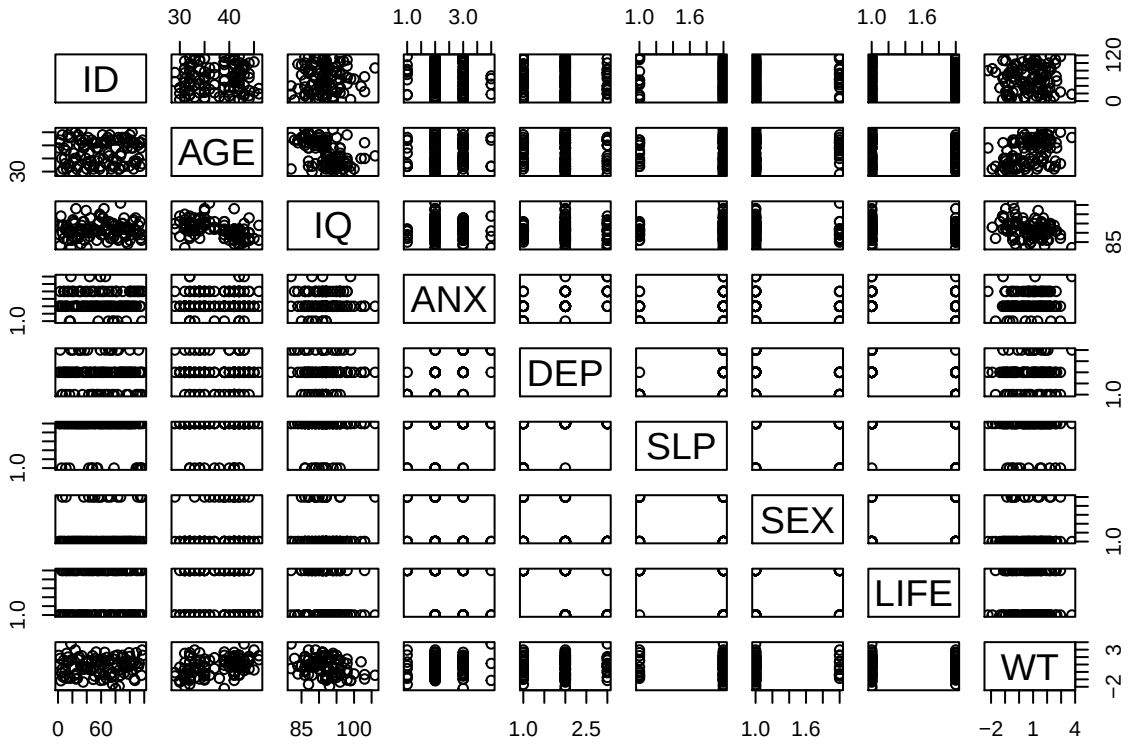
```
## data:  IQ and WT
## t = -3.0192, df = 98, p-value = 0.003231
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.4616804 -0.1010899
## sample estimates:
##       cor
## -0.2917158
```

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)
```

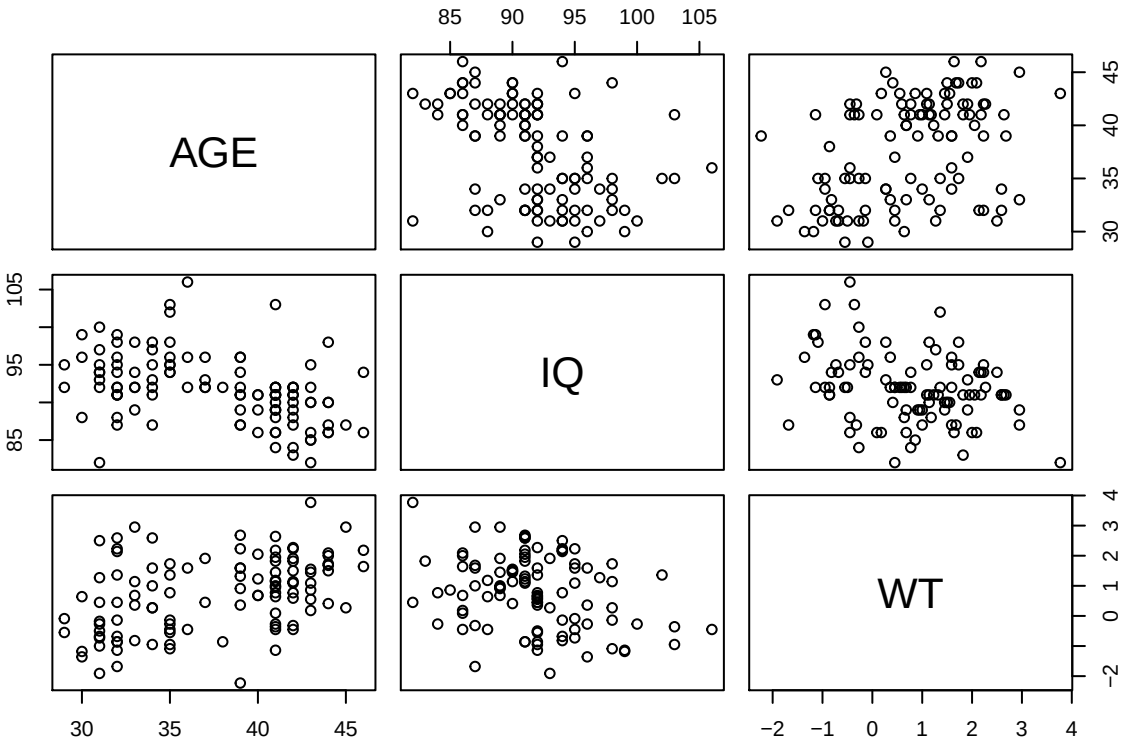


The output can be a little confusing particularly if it includes categorical or record identifying variables. To avoid this we can create a new object that contains only the columns we are interested in using the column binding `cbind()` function:

```
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:

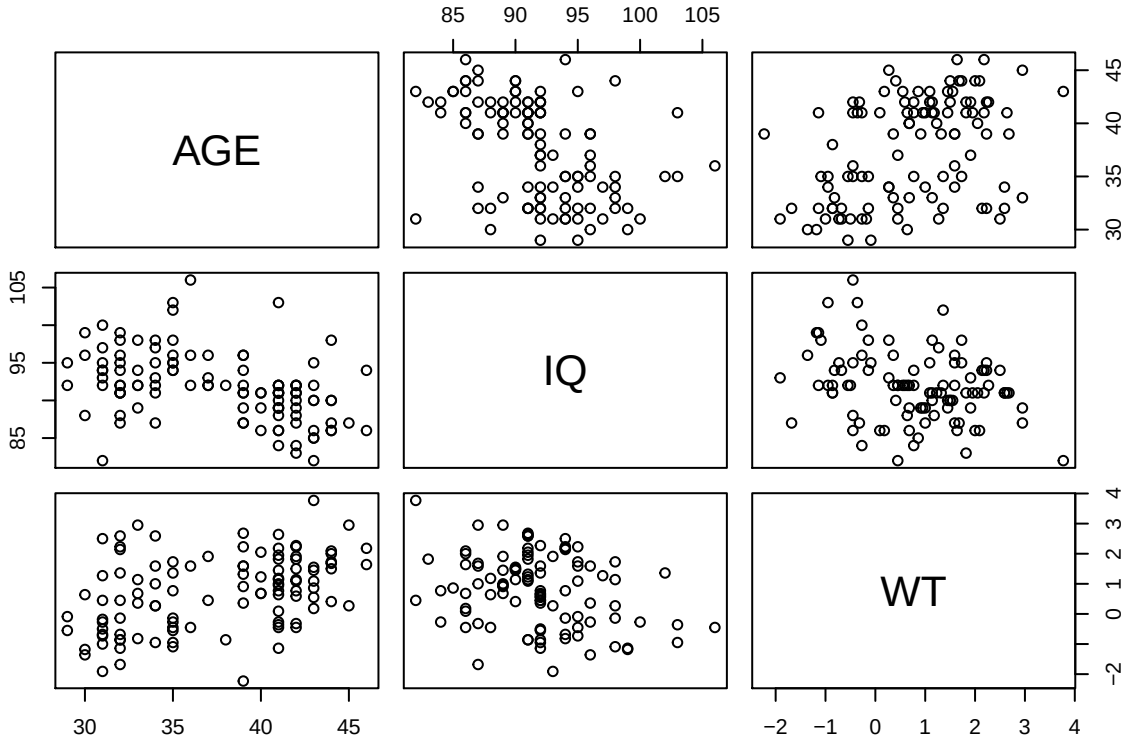
```
rm(newfem)
```

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
```

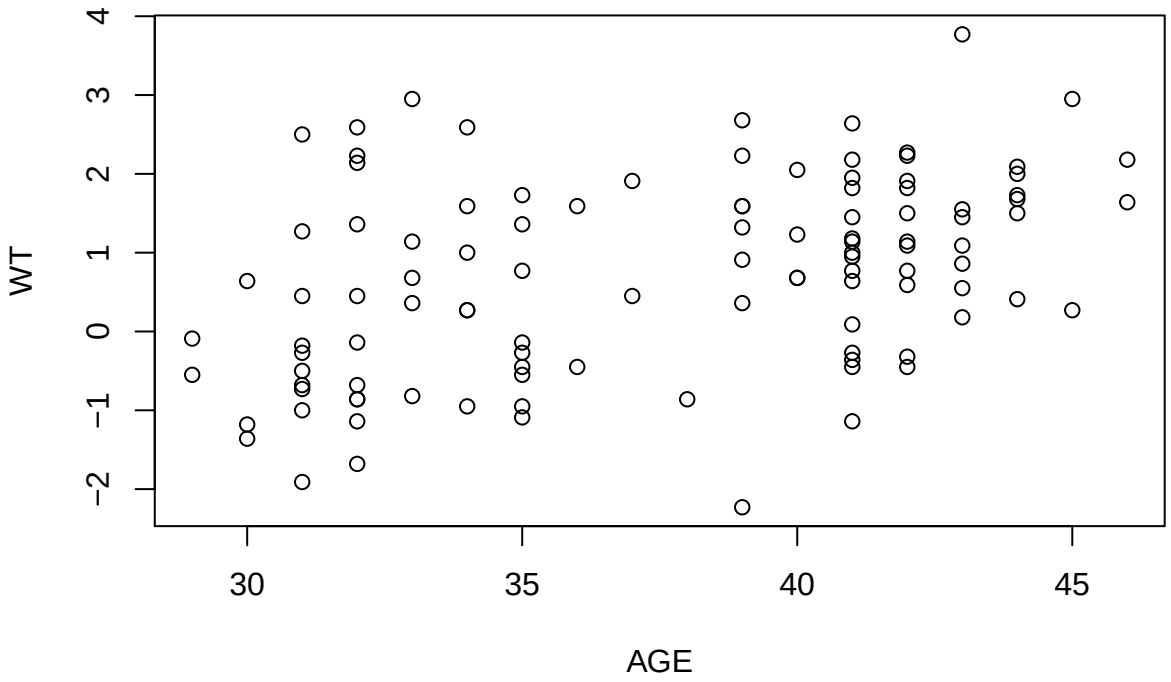
```
pairs(cbind(AGE, IQ, WT))
```



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:

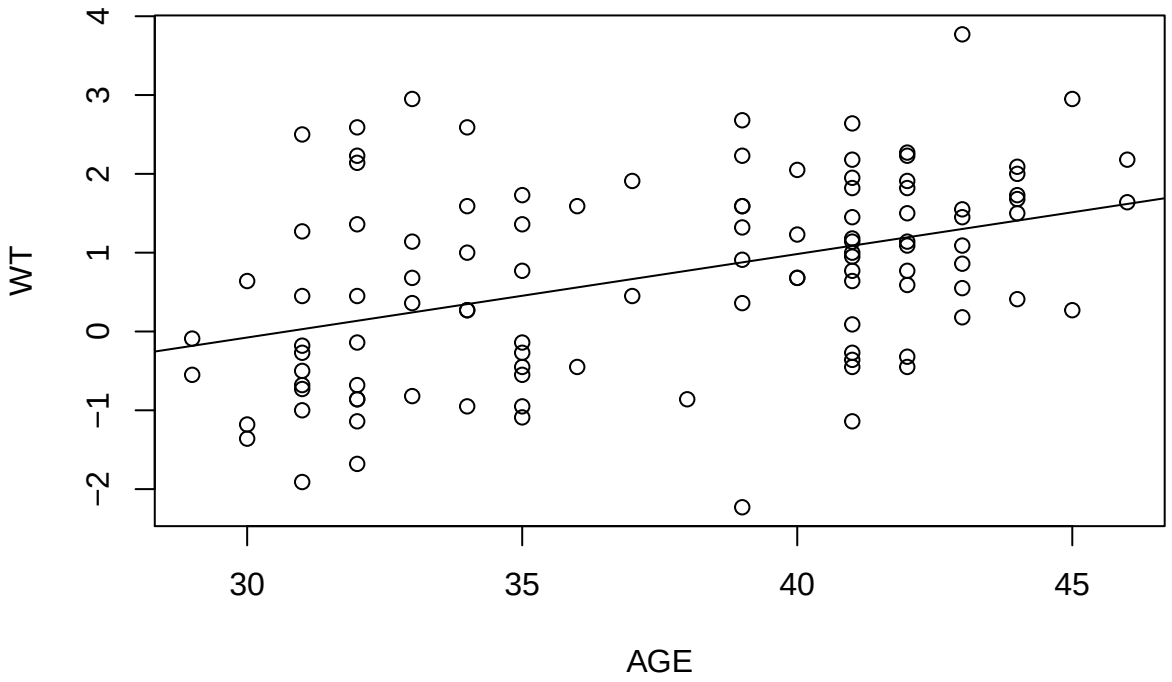
```
fem.lm <- lm(WT ~ AGE)
```

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
```

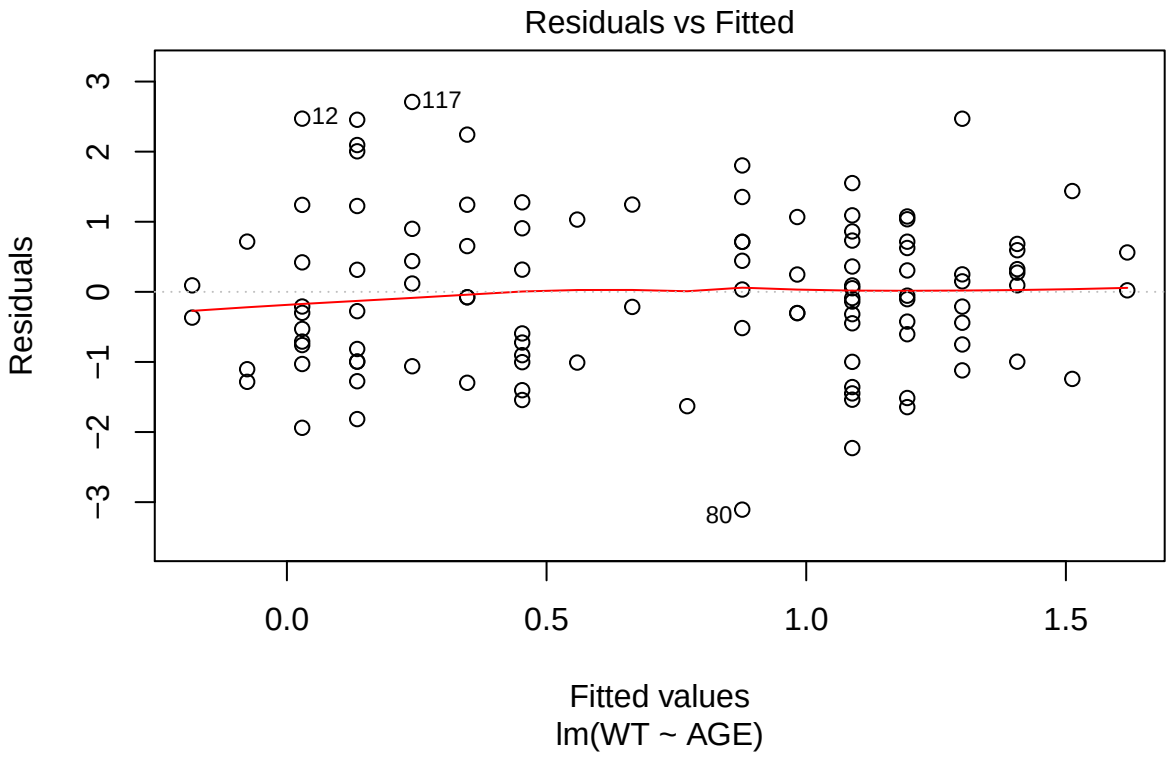
```
plot(AGE, WT)
abline(fem.lm)
```

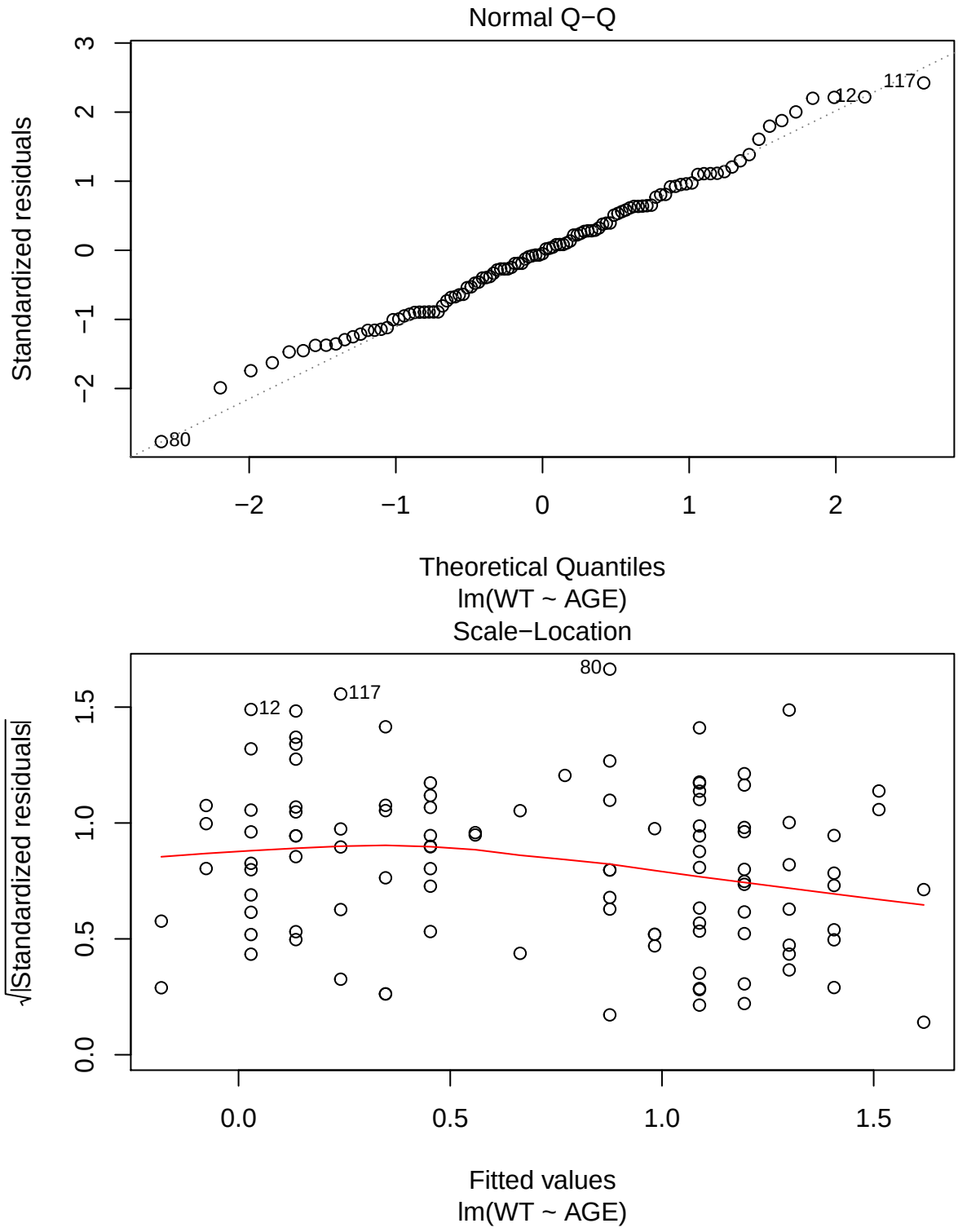


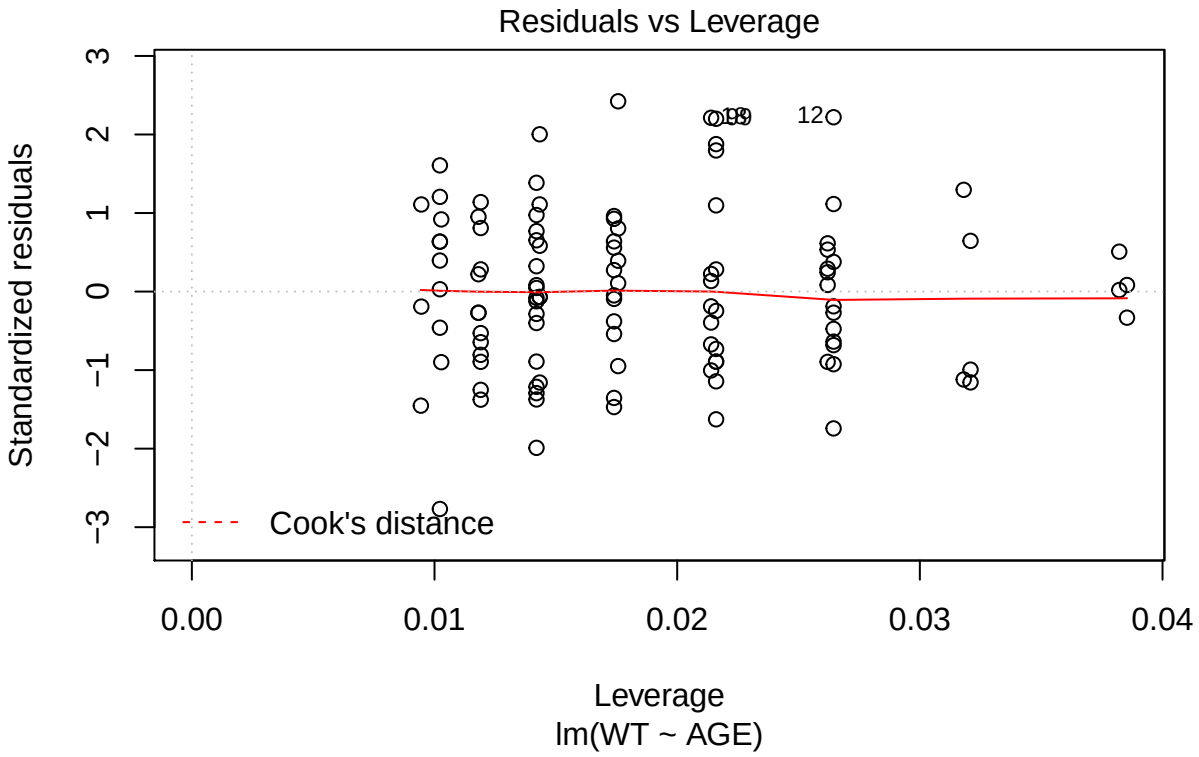
In this case we are passing the intercept and slope information held in the `fem.lm` object to the `abline()` function which draws a regression line. The `abline()` function adds to an existing plot. This means that you need to keep the scatter plot of `AGE` and `WT` open before issuing the `abline()`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




Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(WT ~ AGE)
80
12
117
Scale−Location
√|Standardized residuals|
Fitted values
lm(WT ~ AGE)

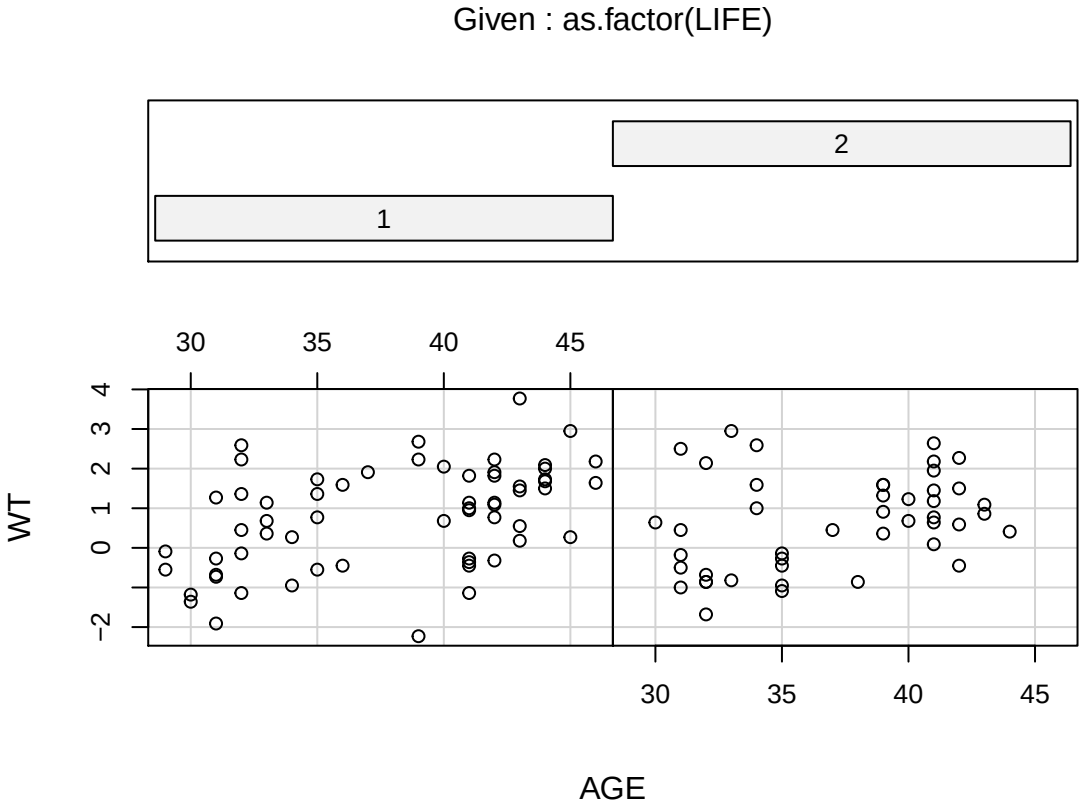


Objects created by the `lm()` function (or any of the modelling functions) can use up a lot of memory so we should remove them when we no longer need them:

```
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:

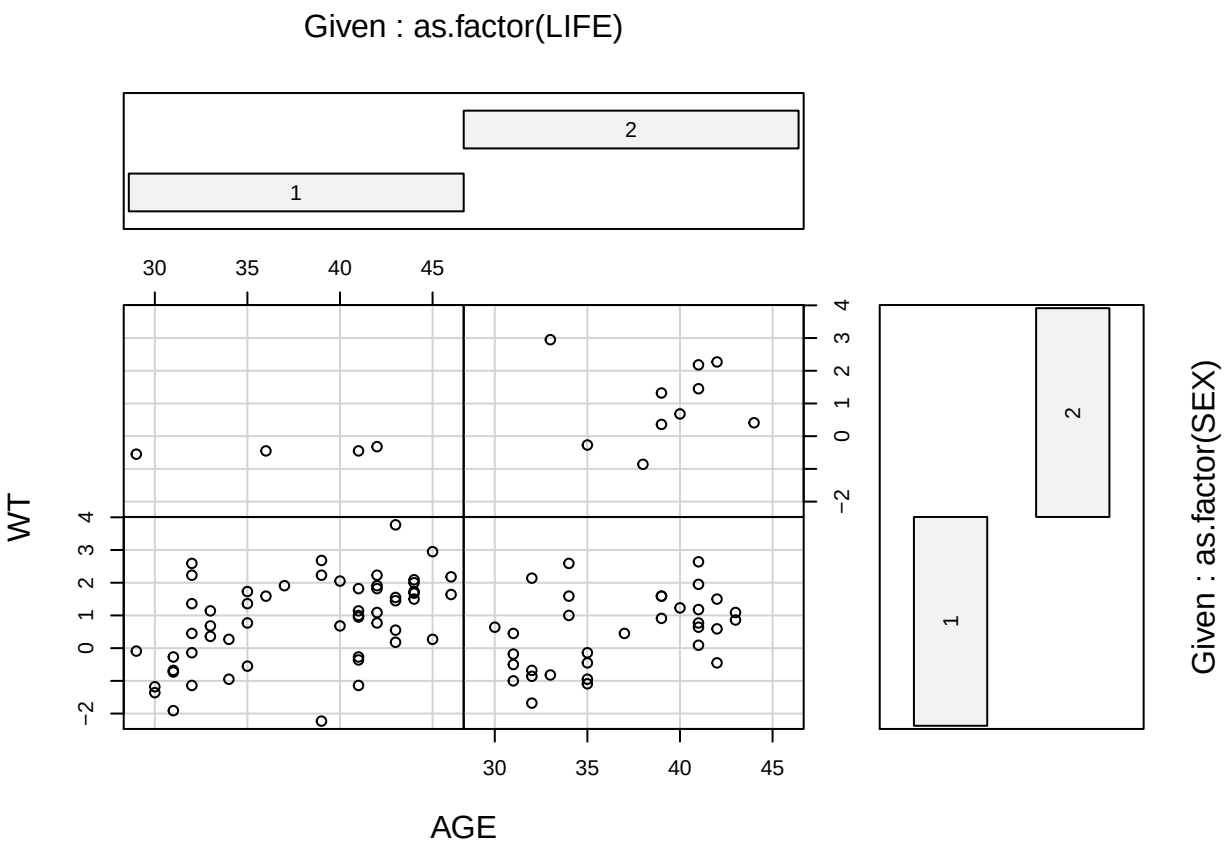
```
coplot(WT ~ AGE | as.factor(LIFE))
```




```
##
##  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 = 10):
```

```
##
##     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
```

We seldom need to report estimates and confidence intervals to more than two decimal places. We can use the `round()` function to remove the excess digits:

```
round(lreg.or, digits = 2)
```

```
##                or  lci    uci
## (Intercept)  0.14 0.04   0.46
## EGGS        14.00 3.33  58.94
## PASTA        5.28 1.02  27.28
## LETTUCE     24.42 2.56 233.40
```

We have now gone through all the necessary calculations step-by-step but it would be nice to have a function that did it all for us that we could use whenever we needed to.

First we will create a template for the function:

```
lreg.or <- function(model, digits = 2) {}
```

and then use the `fix()` function to edit the `lreg.or()` function:

```
fix(lreg.or)
```

We can now edit this function to add a calculation of odds ratios and 95% confidence intervals:

```
function(model, digits = 2) {
  lreg.coeffs <- coef(summary(model))
  OR <- exp(lreg.coeffs[ ,1])
  LCI <- exp(lreg.coeffs[ ,1] - 1.96 * lreg.coeffs[ ,2])
  UCI <- exp(lreg.coeffs[ ,1] + 1.96 * lreg.coeffs[ ,2])
  lreg.or <- round(cbind(OR, LCI, UCI), digits = digits)
  lreg.or
}
```

```
lreg.or <- function(model, digits = 2) {
  lreg.coeffs <- coef(summary(model))
  OR <- exp(lreg.coeffs[ ,1])
  LCI <- exp(lreg.coeffs[ ,1] - 1.96 * lreg.coeffs[ ,2])
  UCI <- exp(lreg.coeffs[ ,1] + 1.96 * lreg.coeffs[ ,2])
  lreg.or <- round(cbind(OR, LCI, UCI), digits = digits)
  lreg.or
}
```

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

We can test our function:

```
lreg.or(salex.lreg)
```

Which produces the following output:

```
lreg.or(salex.lreg)
```

```
##                OR  LCI    UCI
## (Intercept)  0.14 0.04   0.46
## EGGS        14.00 3.33  58.94
## PASTA        5.28 1.02  27.28
## LETTUCE     24.42 2.56 233.40
```

The `digits` parameter of the `lreg.or()` function, which has `digits = 2` as its default value, allows us to specify the precision with which the estimates and their confidence intervals are reported:

```
lreg.or(salex.lreg, digits = 4)
```

```
##                  OR    LCI      UCI
## (Intercept)  0.1393 0.0418   0.4647
## EGGS        14.0008 3.3257  58.9423
## PASTA        5.2835 1.0232  27.2832
## LETTUCE     24.4247 2.5560 233.4018
```

Before we continue, it is probably a good idea to save this function for later use:

```
save(lreg.or, file = "lregor.r")
```

Which can be reloaded whenever it is needed:

```
load("lregor.r")
```

An alternative to using logistic regression with data that contains associations that may be due to confounding is to use stratified analysis (i.e. *Mantel-Haenszel* techniques). With several potential confounders, a stratified analysis results in the analysis of many tables which can be difficult to interpret. For example, four potential confounders, each with two levels would produce sixteen tables. In such situations, logistic regression might be a better approach. In order to illustrate Mantel-Haenszel techniques in `R` we will work with a simpler dataset.

On Saturday, 21st April 1990, a luncheon was held in the home of Jean Bateman. There was a total of forty-five guests which included thirty-five members of the Department of Epidemiology and Population Sciences at the London School of Hygiene and Tropical Medicine. On Sunday morning, 22nd April 1990, Jean awoke with symptoms of gastrointestinal illness; her husband awoke with similar symptoms. The possibility of an outbreak related to the luncheon was strengthened when several of the guests telephoned Jean on Sunday and reported illness. On Monday, 23rd April 1990, there was an unusually large number of department members absent from work and reporting illness. Data from this outbreak is stored in the file `bateman.dat`.

The variables in the file `bateman.dat` are:

ILL	Ill?
CHEESE	Cheddar cheese
CRABDIP	Crab dip
CRISPS	Crisps
BREAD	French bread
CHICKEN	Chicken (roasted, served warm)
RICE	Rice (boiled, served warm)
CAESAR	Caesar salad
TOMATO	Tomato salad
ICECREAM	Vanilla ice-cream
CAKE	Chocolate cake
JUICE	Orange juice
WINE	White wine
COFFEE	Coffee

Data is available for all forty-five guests at the luncheon. All of the variables are coded 1=yes, 2=no. Retrieve and attach the `bateman` dataset in `R`:

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

```
##    ILL CHEESE CRABDIP CRISPS BREAD CHICKEN RICE CAESAR TOMATO ICECREAM
## 1    1      1       1      1     2       1    1      1      1         1
## 2    2      1       1      1     2       1    2      2      2         1
## 3    1      2       2      1     2       1    2      1      2         1
## 4    1      1       2      1     1       1    2      1      2         1
## 5    1      1       1      1     2       1    1      1      1         2
## 6    1      1       1      1     1       1    2      1      1         2
## 7    1      2       1      1     2       1    1      1      1         1
## 8    2      1       1      1     2       1    1      2      1         1
## 9    2      1       1      1     2       1    1      2      1         1
## 10   2      2       1      1     2       1    2      2      2         1
## 11   1      1       2      1     1       1    1      1      1         1
## 12   1      1       1      1     1       1    1      1      1         1
## 13   2      2       1      1     2       1    1      2      2         1
## 14   1      2       1      1     1       1    1      1      1         1
## 15   1      1       1      1     2       2    1      1      1         2
## 16   1      2       2      2     2       1    1      1      1         1
## 17   2      1       2      1     1       1    1      2      2         1
## 18   1      2       1      1     2       1    1      1      1         1
## 19   1      1       2      2     1       1    1      2      1         1
## 20   2      2       2      2     2       2    2      2      2         2
## 21   2      1       2      2     1       2    1      1      2         2
```

```
## 22   2      2       2      2      2       2     2      2      2       1
## 23   2      2       2      2      2       2     2      2      2       1
## 24   1      2       1      1      2       1     1      1      2       1
## 25   1      1       2      2      1       1     1      1      1       1
## 26   2      2       1      1      1       1     1      2      2       2
## 27   2      2       1      1      1       1     1      2      2       2
## 28   1      2       1      2      2       1     1      2      2       1
## 29   1      1       2      2      1       1     1      2      2       1
## 30   1      2       1      1      2       1     1      1      1       1
## 31   1      2       1      1      2       1     1      1      1       1
## 32   1      1       2      2      2       1     1      1      1       2
## 33   2      1       2      1      1       1     1      1      1       1
## 34   1      2       1      1      2       1     1      1      1       2
## 35   1      1       2      1      2       1     1      1      1       2
## 36   2      1       2      1      1       2     1      1      1       2
## 37   1      2       1      1      2       1     1      1      1       1
## 38   1      1       2      2      2       1     2      1      1       1
## 39   2      2       1      1      1       1     1      1      1       2
## 40   1      1       1      1      2       1     2      1      1       2
## 41   2      2       1      1      1       1     2      1      1       2
## 42   1      1       2      2      1       2     2      1      1       1
## 43   1      2       1      1      2       1     2      1      1       2
## 44   1      2       1      1      2       2     1      1      1       2
## 45   1      2       1      1      2       2     1      1      1       1
##    CAKE JUICE WINE COFFEE
## 1     1     1    1      1
## 2     1     1    1      2
## 3     1     2    1      2
## 4     1     2    1      2
## 5     1     1    1      1
## 6     1     1    2      2
## 7     1     2    1      1
## 8     1     2    1      1
## 9     1     2    1      1
## 10    2     1    2      1
## 11    1     1    1      2
## 12    1     2    1      1
## 13    1     2    1      1
## 14    2     2    1      1
## 15    1     1    1      1
## 16    1     2    1      2
## 17    2     2    1      2
## 18    1     2    1      1
## 19    1     2    1      1
## 20    1     2    2      1
```

```
## 21    2     2    1      1
## 22    1     2    1      2
## 23    1     2    1      2
## 24    2     2    1      1
## 25    2     2    1      1
## 26    1     1    1      1
## 27    1     1    1      1
## 28    2     1    2      1
## 29    1     1    2      1
## 30    1     2    2      1
## 31    2     2    2      1
## 32    1     2    2      1
## 33    1     2    2      1
## 34    1     2    1      2
## 35    1     2    1      1
## 36    2     2    1      1
## 37    2     2    1      2
## 38    1     2    1      2
## 39    1     1    2      2
## 40    1     2    1      1
## 41    2     2    1      1
## 42    2     2    1      2
## 43    1     2    1      2
## 44    1     2    1      2
## 45    1     1    2      2
```

```
attach(bateman)
```

```
## The following objects are masked from salex (pos = 3):
##
##     CRISPS, ILL, RICE, TOMATO

## The following objects are masked from salex (pos = 11):
##
##     CRISPS, ILL, RICE, TOMATO
```

We will use our `tab2by2()` function to analyse this data. Retrieve this function:

```
load("tab2by2.r")
```

Use the `tab2by2()` function to analyse the data:

```
tab2by2(CHEESE, ILL)
```

```
##
##          outcome
## exposure  1  2
##        1 15  7
```

```
##        2 14  9
##
## Relative Risk     : 1.12013
## 95% CI            : 0.7253229 1.729838
##
## Sample Odds Ratio : 1.377551
## 95% CI            : 0.4037553 4.699992
##
## MLE Odds Ratio    : 1.367743
## 95% CI             : 0.3427732 5.649399
```

```
tab2by2(CRABDIP, ILL)
```

```
##
##         outcome
## exposure  1  2
##        1 18  9
##        2 11  7
##
## Relative Risk     : 1.090909
## 95% CI            : 0.6921784 1.719329
##
## Sample Odds Ratio : 1.272727
## 95% CI            : 0.3682028 4.3993
##
## MLE Odds Ratio    : 1.265848
## 95% CI             : 0.3042941 5.188297
```

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

```
##
##         outcome
## exposure  1  2
##        1 21 12
##        2  8  4
##
## Relative Risk     : 0.9545455
## 95% CI            : 0.5930168 1.536478
##
## Sample Odds Ratio : 0.875
## 95% CI            : 0.2170373 3.527619
##
## MLE Odds Ratio    : 0.8775841
## 95% CI             : 0.1587568 4.184763
```

```
tab2by2(BREAD, ILL)
```

```
##
##          outcome
## exposure  1  2
##        1  9  8
##        2 20  8
##
## Relative Risk     : 0.7411765
## 95% CI            : 0.4469843 1.228997
##
## Sample Odds Ratio : 0.45
## 95% CI            : 0.1280647 1.581232
##
## MLE Odds Ratio    : 0.4584416
## 95% CI             : 0.1072622 1.897017
```

```
tab2by2(CHICKEN, ILL)
```

```
##
##          outcome
## exposure  1  2
##        1 25 11
##        2  4  5
##
## Relative Risk     : 1.5625
## 95% CI            : 0.7293337 3.347448
##
## Sample Odds Ratio : 2.840909
## 95% CI            : 0.637796 12.65415
##
## MLE Odds Ratio    : 2.76979
## 95% CI             : 0.4912167 16.93409
```

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

```
##
##          outcome
## exposure  1  2
##        1 22 10
##        2  7  6
##
## Relative Risk     : 1.276786
## 95% CI            : 0.7330759 2.223756
##
## Sample Odds Ratio : 1.885714
```

```
## 95% CI             : 0.5027038 7.073586
##
## MLE Odds Ratio     : 1.85813
## 95% CI              : 0.4026256 8.531602
```

```
tab2by2(CAESAR, ILL)
```

```
##
##          outcome
## exposure  1  2
##        1 26  5
##        2  3 11
##
## Relative Risk      : 3.913978
## 95% CI             : 1.418617 10.7987
##
## Sample Odds Ratio : 19.06667
## 95% CI             : 3.866585 94.02038
##
## MLE Odds Ratio     : 17.33517
## 95% CI              : 3.179027 133.7994
```

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

```
##
##          outcome
## exposure  1  2
##        1 24  6
##        2  5 10
##
## Relative Risk      : 2.4
## 95% CI             : 1.14769 5.018775
##
## Sample Odds Ratio : 8
## 95% CI             : 1.97785 32.35836
##
## MLE Odds Ratio     : 7.553116
## 95% CI              : 1.642249 41.02567
```

```
tab2by2(ICECREAM, ILL)
```

```
##
##          outcome
## exposure  1  2
##        1 20  9
##        2  9  7
##
```

```
## Relative Risk      : 1.226054
## 95% CI             : 0.7463643 2.01404
##
## Sample Odds Ratio : 1.728395
## 95% CI             : 0.4889138 6.110177
##
## MLE Odds Ratio     : 1.7069
## 95% CI              : 0.4021245 7.255001
```

```
tab2by2(CAKE, ILL)
```

```
##
##          outcome
## exposure  1  2
##        1 22 11
##        2  7  5
##
## Relative Risk      : 1.142857
## 95% CI             : 0.6689315 1.95255
##
## Sample Odds Ratio : 1.428571
## 95% CI             : 0.3678242 5.548347
##
## MLE Odds Ratio     : 1.416945
## 95% CI              : 0.2847257 6.685098
```

```
tab2by2(JUICE, ILL)
```

```
##
##          outcome
## exposure  1  2
##        1  8  5
##        2 21 11
##
## Relative Risk      : 0.9377289
## 95% CI             : 0.5701453 1.542301
##
## Sample Odds Ratio : 0.8380952
## 95% CI             : 0.2206785 3.182927
##
## MLE Odds Ratio     : 0.8414367
## 95% CI              : 0.185464 4.101313
```

```
tab2by2(WINE, ILL)
```

```
##
##          outcome
```

```
## exposure  1  2
##        1 22 12
##        2  7  4
##
## Relative Risk     : 1.016807
## 95% CI            : 0.6099343 1.695094
##
## Sample Odds Ratio : 1.047619
## 95% CI            : 0.2543383 4.315141
##
## MLE Odds Ratio    : 1.046515
## 95% CI             : 0.1855742 5.186546
```

```
tab2by2(COFFEE, ILL)
```

```
##
##         outcome
## exposure  1  2
##        1 17 11
##        2 12  5
##
## Relative Risk     : 0.860119
## 95% CI            : 0.5607997 1.319196
##
## Sample Odds Ratio : 0.6439394
## 95% CI            : 0.1772875 2.338901
##
## MLE Odds Ratio    : 0.6502015
## 95% CI             : 0.1388979 2.729586
```

Two variables (CAESAR and TOMATO) are associated with ILL.

These two variables are also associated with each other:

```
tab2by2(CAESAR, TOMATO)
```

```
##
##         outcome
## exposure  1  2
##        1 27  4
##        2  3 11
##
## Relative Risk     : 4.064516
## 95% CI            : 1.477162 11.1838
##
## Sample Odds Ratio : 24.75
## 95% CI            : 4.738936 129.2616
```

```
##
## MLE Odds Ratio    : 22.10962
## 95% CI             : 3.850174 183.4671
```

```
chisq.test(table(CAESAR, TOMATO))
```

```
## Warning in chisq.test(table(CAESAR, TOMATO)): Chi-squared approximation may
## be incorrect
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  table(CAESAR, TOMATO)
## X-squared = 15.877, df = 1, p-value = 6.759e-05
```

```
fisher.test(table(CAESAR, TOMATO))
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  table(CAESAR, TOMATO)
## p-value = 3.442e-05
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##    3.850174 183.467108
## sample estimates:
## odds ratio
##   22.10962
```

This suggests the potential for one of these associations to be due to confounding. We can perform a simple stratified analysis using the `table()` function:

```
table(CAESAR, ILL, TOMATO)
```

```
## , , TOMATO = 1
##
##       ILL
## CAESAR  1  2
##      1 23  4
##      2  1  2
##
## , , TOMATO = 2
##
##       ILL
## CAESAR  1  2
##      1  3  1
##      2  2  9
```

```
table(TOMATO, ILL, CAESAR)
```

```
## , , CAESAR = 1
##
##       ILL
## TOMATO  1  2
##      1 23  4
##      2  3  1
##
## , , CAESAR = 2
##
##       ILL
## TOMATO  1  2
##      1  1  2
##      2  2  9
```

It would be useful to calculate odds ratios for each stratum. We can define a simple function to calculate an odds ratio from a two-by-two table:

```
or <- function(x) {(x[1,1] / x[1,2]) / (x[2,1] / x[2,2])}
```

We can use `apply()` to apply the `or()` function to the two-by-two table in each stratum:

```
tabC <- table(CAESAR, ILL, TOMATO)
apply(tabC, 3, or)
```

```
##    1    2
## 11.5 13.5
```

```
tabT <- table(TOMATO, ILL, CAESAR)
apply(tabT, 3, or)
```

```
##        1        2
## 1.916667 2.250000
```

The 3 instructs the `apply()` function to apply the `or()` function to the third dimension of the table objects (i.e. levels of the potential confounder in `tabC` and `tabT`).

The `mantelhaen.test()` function performs the stratified analysis:

```
mantelhaen.test(tabC)
```

```
##
##  Mantel-Haenszel chi-squared test with continuity correction
##
## data:  tabC
## Mantel-Haenszel X-squared = 5.752, df = 1, p-value = 0.01647
## alternative hypothesis: true common odds ratio is not equal to 1
## 95 percent confidence interval:
```

```
##   1.878994 83.156212
## sample estimates:
## common odds ratio
##              12.5
```

```
mantelhaen.test(tabT)
```

```
##
##  Mantel-Haenszel chi-squared test with continuity correction
##
## data:  tabT
## Mantel-Haenszel X-squared = 0.049144, df = 1, p-value = 0.8246
## alternative hypothesis: true common odds ratio is not equal to 1
## 95 percent confidence interval:
##   0.3156862 13.4192331
## sample estimates:
## common odds ratio
##          2.058219
```

It is likely that CAESAR salad was a vehicle of food-poisoning, and that TOMATO salad was not a vehicle of food-poisoning. Many of those at the luncheon ate both CAESAR salad and TOMATO salad. CAESAR confounded the relationship between TOMATO and ILL. This resulted in a spurious association between TOMATO and ILL.

It only makes sense to calculate a common odds ratio in the absence of interaction. We can check for interaction 'by eye' by examining and comparing the odds ratios for each stratum as we did above.

There does appear to be an interaction between CAESAR, WINE, and ILL:

```
tabW <- table(CAESAR, ILL, WINE)
apply(tabW, 3, or)
```

```
##    1    2
## 63.0  2.5
```

Woolf's test for interaction (also known as *Woolf's test for the homogeneity of odds ratios*) provides a formal test for interaction.

R does not provide a function to perform *Woolf's test* for the homogeneity of odds ratios but it is possible to write a function to perform this test.

First we will create a template for the function:

```
woolf.test <- function(x) {}
```

And then use the fix() function to edit the woolf.test() function:

```
fix(woolf.test)
```

We can now edit this function to make it do something useful:

```
function(x) {
  x <- x + 0.5
  k <- dim(x)[3]
  or <- apply(x, 3, function(x)
              {(x[1, 1] / x[1, 2]) / (x[2, 1] / x[2, 2])})
  w <- apply(x, 3, function(x) {1 / sum(1 / x)})
  chi.sq <- sum(w * (log(or) - weighted.mean(log(or), w))^2)
  p <- pchisq(chi.sq, df = k - 1, lower.tail = FALSE)
  cat("\nWoolf's X2 :", chi.sq,
      "\np-value    :", p, "\n")
}
```

Once you have made the changes shown above, check your work, save the file, and quit the editor. We can use the `woolf.test()` function to test for a three-way interaction between `CAESAR`, `WINE`, and `ILL`:

```
woolf.test(tabW)
```

Which returns:

```
woolf.test(tabW)
```

```
##
## Woolf's X2 : 3.319492
## p-value    : 0.06846297
```

Which is weak evidence of an interaction.

We should test for interaction between `CAESAR`, `TOMATO`, and `ILL` before accepting the results reported by the `mantelhaen.test()` function:

```
woolf.test(tabC)
```

```
##
## Woolf's X2 : 0.0001233783
## p-value    : 0.9911376
```

We can repeat this analysis using logistic regression.

We need to change the coding of the variables to 0 and 1 before specifying the model:

```
detach(bateman)
bateman <- 2 - bateman
bateman
bateman.lreg <- glm(formula = ILL ~ CAESAR + TOMATO,
                    family = binomial(logit), data = bateman)
summary(bateman.lreg)
bateman.lreg <- update(bateman.lreg, . ~ . - TOMATO)
summary(bateman.lreg)
```

```
##    ILL CHEESE CRABDIP CRISPS BREAD CHICKEN RICE CAESAR TOMATO ICECREAM
## 1    1      1       1      1     0       1    1      1      1         1
## 2    0      1       1      1     0       1    0      0      0         1
## 3    1      0       0      1     0       1    0      1      0         1
## 4    1      1       0      1     1       1    0      1      0         1
## 5    1      1       1      1     0       1    1      1      1         0
## 6    1      1       1      1     1       1    0      1      1         0
## 7    1      0       1      1     0       1    1      1      1         1
## 8    0      1       1      1     0       1    1      0      1         1
## 9    0      1       1      1     0       1    1      0      1         1
## 10   0      0       1      1     0       1    0      0      0         1
## 11   1      1       0      1     1       1    1      1      1         1
## 12   1      1       1      1     1       1    1      1      1         1
## 13   0      0       1      1     0       1    1      0      0         1
## 14   1      0       1      1     1       1    1      1      1         1
## 15   1      1       1      1     0       0    1      1      1         0
## 16   1      0       0      0     0       1    1      1      1         1
## 17   0      1       0      1     1       1    1      0      0         1
## 18   1      0       1      1     0       1    1      1      1         1
## 19   1      1       0      0     1       1    1      0      1         1
## 20   0      0       0      0     0       0    0      0      0         0
## 21   0      1       0      0     1       0    1      1      0         0
## 22   0      0       0      0     0       0    0      0      0         1
## 23   0      0       0      0     0       0    0      0      0         1
## 24   1      0       1      1     0       1    1      1      0         1
## 25   1      1       0      0     1       1    1      1      1         1
## 26   0      0       1      1     1       1    1      0      0         0
## 27   0      0       1      1     1       1    1      0      0         0
## 28   1      0       1      0     0       1    1      0      0         1
## 29   1      1       0      0     1       1    1      0      0         1
## 30   1      0       1      1     0       1    1      1      1         1
## 31   1      0       1      1     0       1    1      1      1         1
## 32   1      1       0      0     0       1    1      1      1         0
## 33   0      1       0      1     1       1    1      1      1         1
## 34   1      0       1      1     0       1    1      1      1         0
## 35   1      1       0      1     0       1    1      1      1         0
## 36   0      1       0      1     1       0    1      1      1         0
## 37   1      0       1      1     0       1    1      1      1         1
## 38   1      1       0      0     0       1    0      1      1         1
## 39   0      0       1      1     1       1    1      1      1         0
## 40   1      1       1      1     0       1    0      1      1         0
## 41   0      0       1      1     1       1    0      1      1         0
## 42   1      1       0      0     1       0    0      1      1         1
## 43   1      0       1      1     0       1    0      1      1         0
## 44   1      0       1      1     0       0    1      1      1         0
```

```
## 45   1      0       1      1      0       0    1      1      1        1
##    CAKE JUICE WINE COFFEE
## 1     1     1    1      1
## 2     1     1    1      0
## 3     1     0    1      0
## 4     1     0    1      0
## 5     1     1    1      1
## 6     1     1    0      0
## 7     1     0    1      1
## 8     1     0    1      1
## 9     1     0    1      1
## 10    0     1    0      1
## 11    1     1    1      0
## 12    1     0    1      1
## 13    1     0    1      1
## 14    0     0    1      1
## 15    1     1    1      1
## 16    1     0    1      0
## 17    0     0    1      0
## 18    1     0    1      1
## 19    1     0    1      1
## 20    1     0    0      1
## 21    0     0    1      1
## 22    1     0    1      0
## 23    1     0    1      0
## 24    0     0    1      1
## 25    0     0    1      1
## 26    1     1    1      1
## 27    1     1    1      1
## 28    0     1    0      1
## 29    1     1    0      1
## 30    1     0    0      1
## 31    0     0    0      1
## 32    1     0    0      1
## 33    1     0    0      1
## 34    1     0    1      0
## 35    1     0    1      1
## 36    0     0    1      1
## 37    0     0    1      0
## 38    1     0    1      0
## 39    1     1    0      0
## 40    1     0    1      1
## 41    0     0    1      1
## 42    0     0    1      0
## 43    1     0    1      0
```

```
## 44    1     0    1      0
## 45    1     1    0      0
```

```
##
## Call:
## glm(formula = ILL ~ CAESAR + TOMATO, family = binomial(logit),
##     data = bateman)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.960  -0.641   0.563   0.563   1.835
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.4780     0.7101  -2.082  0.03739 *
## CAESAR        2.5202     0.9653   2.611  0.00904 **
## TOMATO        0.7197     0.9552   0.753  0.45116
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 58.574  on 44  degrees of freedom
## Residual deviance: 41.408  on 42  degrees of freedom
## AIC: 47.408
##
## Number of Fisher Scoring iterations: 4
```

```
##
## Call:
## glm(formula = ILL ~ CAESAR, family = binomial(logit), data = bateman)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9103  -0.6945   0.5931   0.5931   1.7552
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.2993     0.6513  -1.995 0.046066 *
## CAESAR        2.9479     0.8141   3.621 0.000293 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 58.574  on 44  degrees of freedom
```

```
## Residual deviance: 41.940  on 43  degrees of freedom
## AIC: 45.94
##
## Number of Fisher Scoring iterations: 4
```

Interactions are specified using the multiply (___*___) symbol in the model formula:

```
bateman.lreg <- glm(formula = ILL ~ CAESAR + WINE + CAESAR * WINE,
                    family = binomial(logit), data = bateman)
summary(bateman.lreg)
```

```
##
## Call:
## glm(formula = ILL ~ CAESAR + WINE + CAESAR * WINE, family = binomial(logit),
##     data = bateman)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0393  -0.4590   0.5168   0.5168   2.1460
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.092e-15  1.000e+00   0.000   1.0000
## CAESAR       9.163e-01  1.304e+00   0.703   0.4822
## WINE        -2.197e+00  1.453e+00  -1.512   0.1305
## CAESAR:WINE  3.227e+00  1.787e+00   1.806   0.0709 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 58.574  on 44  degrees of freedom
## Residual deviance: 38.508  on 41  degrees of freedom
## AIC: 46.508
##
## Number of Fisher Scoring iterations: 4
```

Before we continue, it is probably a good idea to save the `woolf.test()` function for later use:

```
save(woolf.test, file = "woolf.r")
```

4.1 Matched data

Matching is another way to control for the effects of potential confounding variables. Matching is usually performed during data-collection as part of the design of a study.

In a matched case-control studies, each case is matched with one or more controls which are chosen to have the same values over a set of potential confounding variables. In order to illustrate how matched data may be analysed using tabulation and stratification in R we will start with the simple case of one-to-one matching (i.e. each case has a single matched control):

```
octe <- read.table("octe.dat", header = TRUE)
octe[1:10, ]
```

```
##    ID CASE OC
## 1   1    1  1
## 2   1    2  1
## 3   2    1  1
## 4   2    2  1
## 5   3    1  1
## 6   3    2  1
## 7   4    1  1
## 8   4    2  1
## 9   5    1  1
## 10  5    2  1
```

This data is from a matched case-control study investigating the association between oral contraceptive use and thromboembolism. The cases are 175 women aged between 15 and 44 years admitted to hospital for thromboembolism and discharged alive. The controls are female patients admitted for conditions believed to be unrelated to oral contraceptive use. Cases and controls were matched on age, ethnic group, marital status, parity, income, place of residence, and date of hospitalisation. The variables in the dataset are:

ID	Identifier for the matched sets of cases and controls
CASE	Case (1) or control (2)
OC	Used oral contraceptives in the previous month (1=yes, 2=no)

The dataset consists of 350 records:

```
nrow(octe)
```

```
## [1] 350
```

There are 175 matched sets of cases and controls:

```
length(unique(octe$ID))
```

```
## [1] 175
```

In each matched set of cases and controls there is one case and one control:

```
table(octe$ID, octe$CASE)
```

```
##
##        1 2
##    1   1 1
##    2   1 1
##    3   1 1
##    4   1 1
##    5   1 1
##    6   1 1
##    7   1 1
##    8   1 1
##    9   1 1
##    10  1 1
##    11  1 1
##    12  1 1
##    13  1 1
##    14  1 1
##    15  1 1
##    16  1 1
##    17  1 1
##    18  1 1
##    19  1 1
##    20  1 1
##    21  1 1
##    22  1 1
##    23  1 1
##    24  1 1
##    25  1 1
##    26  1 1
##    27  1 1
##    28  1 1
##    29  1 1
##    30  1 1
##    31  1 1
##    32  1 1
##    33  1 1
##    34  1 1
##    35  1 1
##    36  1 1
##    37  1 1
##    38  1 1
##    39  1 1
```

```
##   40  1 1
##   41  1 1
##   42  1 1
##   43  1 1
##   44  1 1
##   45  1 1
##   46  1 1
##   47  1 1
##   48  1 1
##   49  1 1
##   50  1 1
##   51  1 1
##   52  1 1
##   53  1 1
##   54  1 1
##   55  1 1
##   56  1 1
##   57  1 1
##   58  1 1
##   59  1 1
##   60  1 1
##   61  1 1
##   62  1 1
##   63  1 1
##   64  1 1
##   65  1 1
##   66  1 1
##   67  1 1
##   68  1 1
##   69  1 1
##   70  1 1
##   71  1 1
##   72  1 1
##   73  1 1
##   74  1 1
##   75  1 1
##   76  1 1
##   77  1 1
##   78  1 1
##   79  1 1
##   80  1 1
##   81  1 1
##   82  1 1
##   83  1 1
##   84  1 1
```

```
##   85  1 1
##   86  1 1
##   87  1 1
##   88  1 1
##   89  1 1
##   90  1 1
##   91  1 1
##   92  1 1
##   93  1 1
##   94  1 1
##   95  1 1
##   96  1 1
##   97  1 1
##   98  1 1
##   99  1 1
##   100 1 1
##   101 1 1
##   102 1 1
##   103 1 1
##   104 1 1
##   105 1 1
##   106 1 1
##   107 1 1
##   108 1 1
##   109 1 1
##   110 1 1
##   111 1 1
##   112 1 1
##   113 1 1
##   114 1 1
##   115 1 1
##   116 1 1
##   117 1 1
##   118 1 1
##   119 1 1
##   120 1 1
##   121 1 1
##   122 1 1
##   123 1 1
##   124 1 1
##   125 1 1
##   126 1 1
##   127 1 1
##   128 1 1
##   129 1 1
```

```
##   130 1 1
##   131 1 1
##   132 1 1
##   133 1 1
##   134 1 1
##   135 1 1
##   136 1 1
##   137 1 1
##   138 1 1
##   139 1 1
##   140 1 1
##   141 1 1
##   142 1 1
##   143 1 1
##   144 1 1
##   145 1 1
##   146 1 1
##   147 1 1
##   148 1 1
##   149 1 1
##   150 1 1
##   151 1 1
##   152 1 1
##   153 1 1
##   154 1 1
##   155 1 1
##   156 1 1
##   157 1 1
##   158 1 1
##   159 1 1
##   160 1 1
##   161 1 1
##   162 1 1
##   163 1 1
##   164 1 1
##   165 1 1
##   166 1 1
##   167 1 1
##   168 1 1
##   169 1 1
##   170 1 1
##   171 1 1
##   172 1 1
##   173 1 1
##   174 1 1
```

```
##   175 1 1
```

This data may be analysed using *McNemar's chi-squared test* which use the number of discordant (i.e. relative to exposure) pairs of matched cases and controls.

To find the number of discordant pairs we need to split the dataset into cases and controls:

```
octe.cases <- subset(octe, CASE == 1)
octe.controls <- subset(octe, CASE == 2)
```

Sorting these two datasets (i.e. `octe.cases` and `octe.controls`) by the `ID` variable simplifies the analysis:

```
octe.cases <- octe.cases[order(octe.cases$ID), ]
octe.controls <- octe.controls[order(octe.controls$ID), ]
```

Since the two datasets (i.e. `octe.cases` and `octe.controls`) are now sorted by the `ID` variable we can use the `table()` function to retrieve the number if concordant and discordant pairs and store them in a table object:

```
tab <- table(octe.cases$OC, octe.controls$OC)
tab
```

```
##
##      1  2
##   1 10 57
##   2 13 95
```

This table object (i.e. `tab`) can then be passed to the `mcnemar.test()` function:

```
mcnemar.test(tab)
```

```
##
##  McNemar's Chi-squared test with continuity correction
##
## data:  tab
## McNemar's chi-squared = 26.414, df = 1, p-value = 2.755e-07
```

The `mcnemar.test()` function does not provide an estimate of the odds ratio. This is the ratio of the discordant pairs:

```
r <- tab[1,2]
s <- tab[2,1]
rdp <- r / s
rdp
```

```
## [1] 4.384615
```

A confidence interval can also be calculated:

```
ci.p <- binom.test(r, r + s)$conf.int
ci.rdp <- ci.p / (1 - ci.p)
ci.rdp
```

```
## [1] 2.371377 8.731311
## attr(,"conf.level")
## [1] 0.95
```

This provides a 95% confidence interval. Other (e.g. 99%) confidence intervals can be produced by specifying appropriate values for the `conf.level` parameter of the `binom.test()` function:

```
ci.p <- binom.test(r, r + s, conf.level = 0.99)$conf.int
ci.rdp <- ci.p / (1 - ci.p)
ci.rdp
```

```
## [1]  2.010478 10.949095
## attr(,"conf.level")
## [1] 0.99
```

An alternative way of analysing this data is to use the `mantelhaen.test()` function:

```
tab <- table(octe$OC, octe$CASE, octe$ID)
mantelhaen.test(tab)
```

```
##
##  Mantel-Haenszel chi-squared test with continuity correction
##
## data:  tab
## Mantel-Haenszel X-squared = 26.414, df = 1, p-value = 2.755e-07
## alternative hypothesis: true common odds ratio is not equal to 1
## 95 percent confidence interval:
##  2.400550 8.008521
## sample estimates:
## common odds ratio
##          4.384615
```

The Mantel-Haenszel approach is preferred because it can be used with data from matched case-control studies that match more than one control to each case. Multiple matching is useful when the condition being studied is rare or at the early stages of an outbreak (i.e. when cases are hard to find and controls are easy to find).

We will now work with some data where each case has one or more controls:

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

```
##    ID CASE RBTAMP
## 1   1    1      1
```

```
## 2   1    2      1
## 3   1    2      1
## 4   2    1      1
## 5   2    2      2
## 6   2    2      1
## 7   2    2      1
## 8   3    1      2
## 9   3    2      2
## 10  3    2      1
## 11  4    1      1
## 12  4    2      2
## 13  5    1      1
## 14  5    2      1
## 15  5    2      2
## 16  6    1      1
## 17  6    2      1
## 18  7    1      1
## 19  7    2      2
## 20  7    2      2
## 21  8    1      1
## 22  8    2      1
## 23  8    2      2
## 24  9    1      2
## 25  9    2      1
## 26  9    2      2
## 27  9    2      2
## 28 10    1      1
## 29 10    2      2
## 30 10    2      2
## 31 11    1      1
## 32 11    2      2
## 33 11    2      2
## 34 12    1      1
## 35 12    2      2
## 36 12    2      2
## 37 12    2      2
## 38 13    1      1
## 39 13    2      2
## 40 13    2      2
## 41 14    1      2
## 42 14    2      2
## 43 14    2      2
```

This data is from a matched case-control study investigating the association between the use of different brands of tampon and toxic shock syndrome undertaken during an outbreak.

Only a subset of the original dataset is used here. The variables in the dataset are:

ID	Identifier for the matched sets of cases and controls
CASE	Case (1) or control (2)
RBTAMP	Used Rely brand tampons (1=yes, 2=no)

The dataset consists of forty-three (43) records:

```
nrow(tsstamp)
```

```
## [1] 43
```

There are fourteen (14) matched sets of cases and controls:

```
length(unique(tsstamp$ID))
```

```
## [1] 14
```

Each matched set of cases and controls consists of one case and one or more controls:

```
table(tsstamp$ID, tsstamp$CASE)
```

```
##
##      1 2
##   1  1 2
##   2  1 3
##   3  1 2
##   4  1 1
##   5  1 2
##   6  1 1
##   7  1 2
##   8  1 2
##   9  1 3
##   10 1 2
##   11 1 2
##   12 1 3
##   13 1 2
##   14 1 2
```

The *McNemar's chi-squared test* is not useful for this data as it is limited to the special case of one-to-one matching.

Analysing this data using a simple tabulation such as:

```
fisher.test(table(tsstamp$RBTAMP, tsstamp$CASE))
```

```
##
##  Fisher's Exact Test for Count Data
##
```

```
## data:  table(tsstamp$RBTAMP, tsstamp$CASE)
## p-value = 0.007805
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##   1.542686 53.734756
## sample estimates:
## odds ratio
##   7.709932
```

ignores the matched nature of the data and is, therefore, also not useful for this data.

The matched nature of the data may be accounted by stratifying on the variable that identifies the matched sets of cases and controls (i.e. the `ID` variable) using the `mantelhaen.test()` function:

```
mantelhaen.test(table(tsstamp$RBTAMP, tsstamp$CASE, tsstamp$ID))
```

```
##
##  Mantel-Haenszel chi-squared test with continuity correction
##
## data:  table(tsstamp$RBTAMP, tsstamp$CASE, tsstamp$ID)
## Mantel-Haenszel X-squared = 5.9384, df = 1, p-value = 0.01481
## alternative hypothesis: true common odds ratio is not equal to 1
## 95 percent confidence interval:
##   1.589505 43.191463
## sample estimates:
## common odds ratio
##          8.285714
```

Analysis of several risk factors or adjustment for confounding variables not matched for in the design of a matched case-control study cannot be performed using tabulation-based procedures such as the `McNemar's chi- squared test` and Mantel-Haenszel procedures. In these situations a special form of logistic regression, called `conditional logistic regression`, should be used.

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).

4.2 Summary

- `R` provides functions for many kinds of complex statistical analysis. We have looked at using the generalised linear model `glm()` function to perform logistic regression.

We have looked ar the `mantelhaen.test()` function to perform stratified analyses and the `mantelhaen.test()` and `mcnemar.test()` functions to analyse data from matched case-control studies.

- `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. By building your own functions you can use `R` to build your own statistical analysis system.

Chapter 5

Analysing some data with R

In this exercise we will use the R functions we have already used and the functions we have added to R to analyse a small dataset. First we will start R and retrieve our functions:

```
load("tab2by2.r")
load("lregor.r")
```

And then retrieve and attach the sample dataset:

```
gudhiv <- read.table("gudhiv.dat", header = TRUE, na.strings = "X")
attach(gudhiv)
```

```
## The following objects are masked from gudhiv (pos = 10):
##
##     CIR, GAMBIAN, GUD, HIV, INJ12M, MARRIED, PARTNERS, SEXPRO,
##     TRAVOUT, UTIGC
```

This data is from a cross-sectional study of 435 male patients who presented with sexually transmitted infections at an outpatient clinic in The Gambia between August 1988 and June 1990. Several studies have documented an association between genital ulcer disease (GUD) and HIV infection. A study of Gambian prostitutes documented an association between seropositivity for HIV-2 and antibodies against `Treponema pallidum` (a serological test for syphilis). Prostitutes are not the ideal population for such studies as they may have experienced multiple sexually transmitted infections and it is difficult to quantify the number of times they may have had sex with HIV-2 seropositive customers. A sample of males with sexually transmitted infections is easier to study as they have probably had fewer sexual partners than prostitutes and much less contact with sexually transmitted infection pathogens. In such a sample it is also easier to find subjects and collect data. The variables in the dataset are:

MARRIED	Married (1=yes, 0=no)
GAMBIAN	Gambian Citizen (1=yes, 0=no)
GUD	History of GUD or syphilis (1=yes, 0=no)
UTIGC	History of urethral discharge (1=yes, 0=no)

CIR	Circumcised (1=yes, 0=no)
TRAVOUT	Travelled outside of Gambia and Senegal (1=yes, 0=no)
SEXPRO	Ever had sex with a prostitute (1=yes, 0=no)
INJ12M	Injection in previous 12 months (1=yes, 0=no)
PARTNERS	Sexual partners in previous 12 months (number)
HIV HIV-2	positive serology (1=yes, 0=no)

Data is available for all 435 patients enrolled in the study.

We will start our analysis by examining pairwise associations between the binary exposure variables and the HIV variable using the `tab2by2()` function that we wrote earlier:

```
tab2by2(MARRIED, HIV)
tab2by2(GAMBIAN, HIV)
tab2by2(GUD, HIV)
tab2by2(UTIGC, HIV)
tab2by2(CIR, HIV)
tab2by2(TRAVOUT, HIV)
tab2by2(SEXPRO, HIV)
tab2by2(INJ12M, HIV)
```

```
tab2by2(MARRIED, HIV)
```

```
##
##          outcome
## exposure   0   1
##        0 321  13
##        1  93   8
##
## Relative Risk     : 1.043751
## 95% CI            : 0.9818512 1.109554
##
## Sample Odds Ratio : 2.124069
## 95% CI            : 0.8545749 5.279433
##
## MLE Odds Ratio    : 2.119801
## 95% CI             : 0.7380371 5.714354
```

```
tab2by2(GAMBIAN, HIV)
```

```
##
##          outcome
## exposure   0   1
##        0  73   4
##        1 341  17
##
```

```
## Relative Risk     : 0.9953155
## 95% CI            : 0.9400068 1.053879
##
## Sample Odds Ratio : 0.909824
## 95% CI            : 0.2974059 2.783333
##
## MLE Odds Ratio    : 0.9100104
## 95% CI             : 0.2853202 3.826485
```

```
tab2by2(GUD, HIV)
```

```
##
##          outcome
## exposure   0   1
##        0 339  12
##        1  72   9
##
## Relative Risk     : 1.086538
## 95% CI            : 1.003531 1.176412
##
## Sample Odds Ratio : 3.53125
## 95% CI            : 1.434372 8.693509
##
## MLE Odds Ratio    : 3.517408
## 95% CI             : 1.258556 9.491924
```

```
tab2by2(UTIGC, HIV)
```

```
##
##          outcome
## exposure   0   1
##        0 261  12
##        1 151   9
##
## Relative Risk     : 1.013027
## 95% CI            : 0.9678841 1.060275
##
## Sample Odds Ratio : 1.296358
## 95% CI            : 0.5338453 3.147997
##
## MLE Odds Ratio    : 1.295532
## 95% CI             : 0.4703496 3.438842
```

```
tab2by2(CIR, HIV)
```

```
##
##          outcome
```

```
## exposure   0   1
##        0  10   3
##        1 392  17
##
## Relative Risk     : 0.8025903
## 95% CI            : 0.5955085 1.081682
##
## Sample Odds Ratio : 0.1445578
## 95% CI            : 0.0364195 0.5737851
##
## MLE Odds Ratio    : 0.1460183
## 95% CI             : 0.03322189 0.899754
```

```
tab2by2(TRAVOUT, HIV)
```

```
##
##         outcome
## exposure   0   1
##        0 152   2
##        1 256  19
##
## Relative Risk     : 1.060268
## 95% CI            : 1.02181 1.100173
##
## Sample Odds Ratio : 5.640625
## 95% CI            : 1.295879 24.55218
##
## MLE Odds Ratio    : 5.624226
## 95% CI             : 1.32716 50.45859
```

```
tab2by2(SEXPRO, HIV)
```

```
##
##         outcome
## exposure   0   1
##        0 268  13
##        1 143   8
##
## Relative Risk     : 1.007093
## 95% CI            : 0.9621259 1.054161
##
## Sample Odds Ratio : 1.153308
## 95% CI            : 0.4671083 2.847562
##
## MLE Odds Ratio    : 1.152912
## 95% CI             : 0.4042323 3.083152
```

```
tab2by2(INJ12M, HIV)
```

```
##
##          outcome
## exposure   0   1
##        0 146   7
##        1 268  14
##
## Relative Risk     : 1.004097
## 95% CI            : 0.9610996 1.049018
##
## Sample Odds Ratio : 1.089552
## 95% CI            : 0.4301305 2.759916
##
## MLE Odds Ratio    : 1.089351
## 95% CI             : 0.4006202 3.263814
```

Note that our `tab2by2()` function returns misleading risk ratio estimates and confidence intervals for this dataset. This is because the function expects the `exposure` and `outcome` variables to be ordered with exposure-present and outcome-present as the first category (e.g. 1 = present, 2 = absent). This coding is reversed (i.e. 0 = absent, 1 = present) in the `gudhiv` dataset.

We can produce risk ratio estimates for variables in the `gudhiv` data using the `tab2by2()` function and a simple transformation of the `exposure` and `outcome` variables. For example:

```
tab2by2(2 - GUD, 2 - HIV)
```

```
##
##          outcome
## exposure   1   2
##        1   9  72
##        2  12 339
##
## Relative Risk     : 3.25
## 95% CI            : 1.417411 7.451965
##
## Sample Odds Ratio : 3.53125
## 95% CI            : 1.434372 8.693509
##
## MLE Odds Ratio    : 3.517408
## 95% CI             : 1.258556 9.491924
```

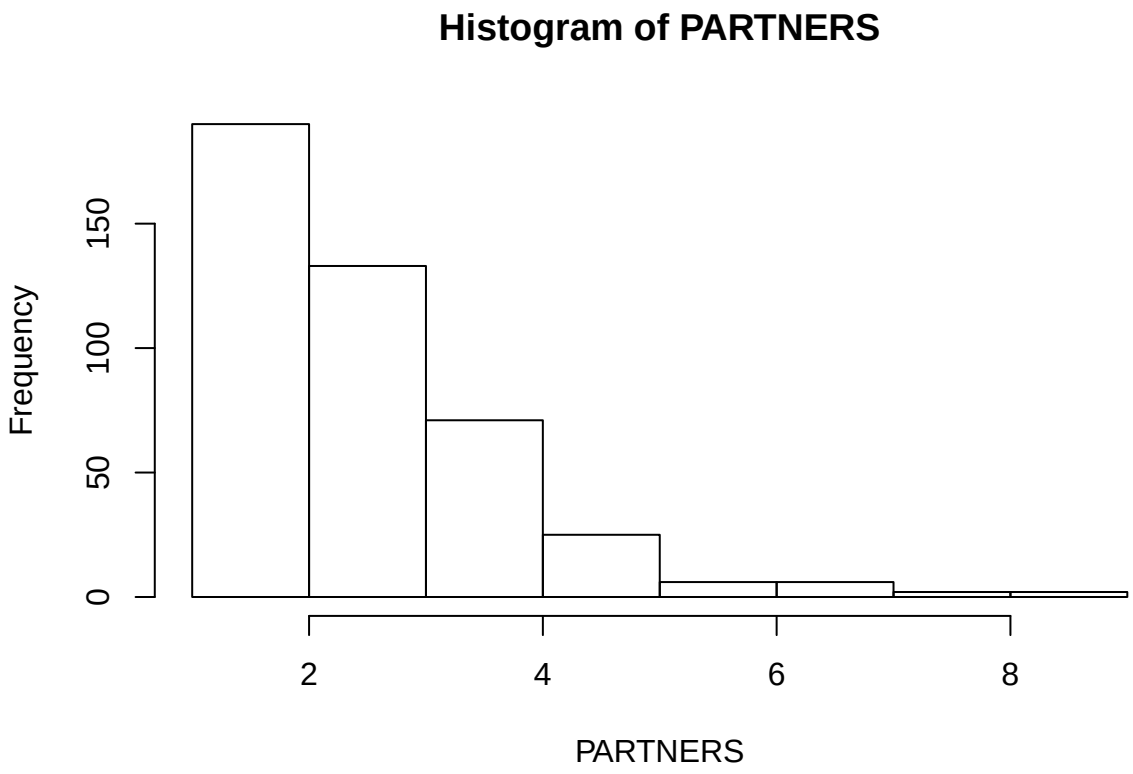
The odds ratio estimates returned by the `tab2by2()` function, with or without this transformation, are correct. The `GUD` and `TRAVOUT` variables are associated with `HIV`.

`PARTNERS` is a continuous variable and we should examine its distribution before doing any-

thing with it:

```
table(PARTNERS)
hist(PARTNERS)
```

```
## PARTNERS
##   1   2   3   4   5   6   7   8   9
##  61 129 133  71  25   6   6   2   2
```



The distribution of `PARTNERS` is severely non-normal. Instead of attempting to transform the variable we will produce summary statistics for each level of the `HIV` variable and perform a non-parametric test:

```
by(PARTNERS, HIV, summary)
kruskal.test(PARTNERS ~ HIV)
```

```
## HIV: 0
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    2.00    3.00    2.72    3.00    8.00
## ------------------------------------------------------------
## HIV: 1
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   4.000   5.000   5.381   7.000   9.000

##
##  Kruskal-Wallis rank sum test
##
```

```
## data:  PARTNERS by HIV
## Kruskal-Wallis chi-squared = 32.036, df = 1, p-value = 1.514e-08
```

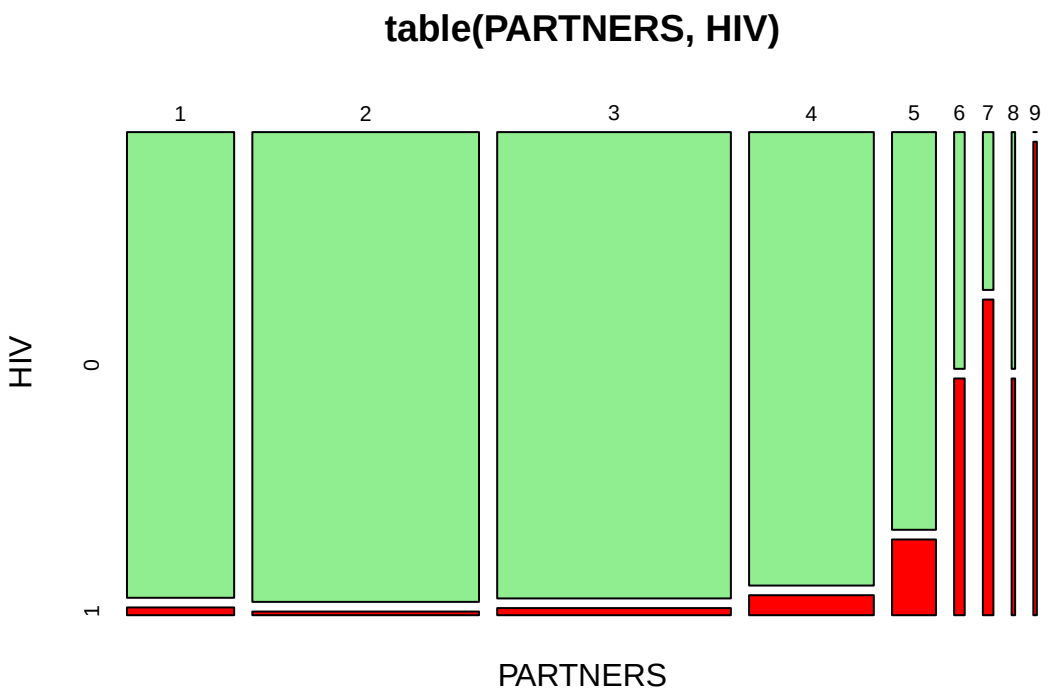
An alternative way of looking at the data is as a tabulation:

```
table(PARTNERS, HIV)
```

```
##          HIV
## PARTNERS   0   1
##        1  60   1
##        2 128   1
##        3 131   2
##        4  68   3
##        5  21   4
##        6   3   3
##        7   2   4
##        8   1   1
##        9   0   2
```

You can use the `plot()` function to represent this table graphically:

```
plot(table(PARTNERS, HIV), color = c("lightgreen", "red"))
```



There appears to be an association between the number of sexual `PARTNERS` in the previous twelve months and positive `HIV` serology. The proportion with positive `HIV` serology increases as the number of sexual partners increases:

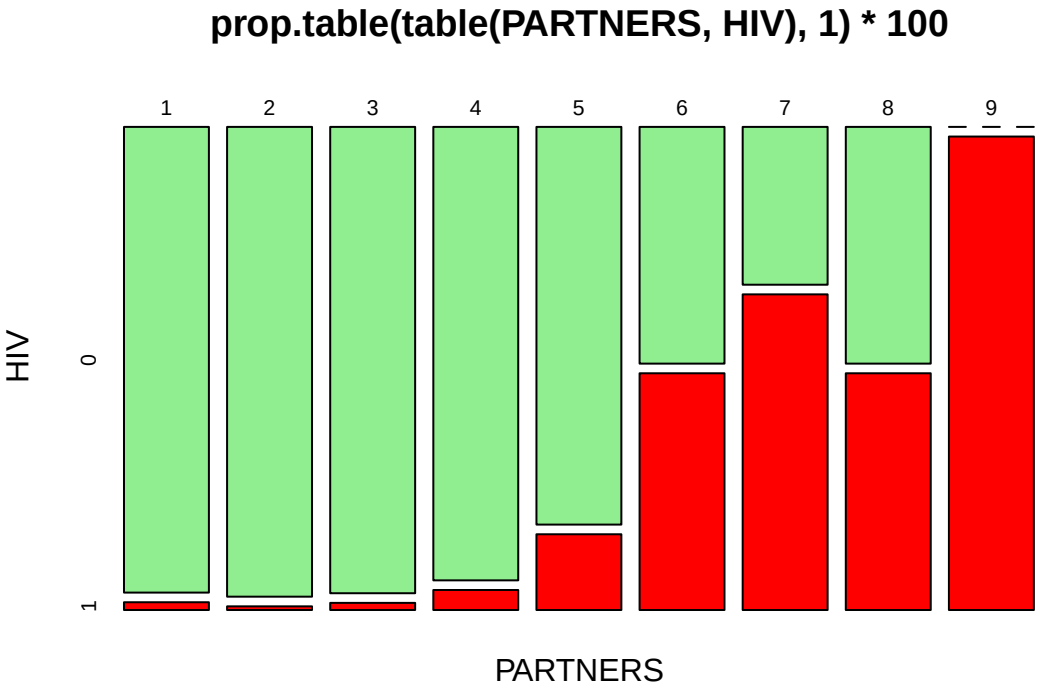
```
prop.table(table(PARTNERS, HIV), 1) * 100
```

```
##          HIV
```

```
## PARTNERS          0           1
##        1  98.3606557   1.6393443
##        2  99.2248062   0.7751938
##        3  98.4962406   1.5037594
##        4  95.7746479   4.2253521
##        5  84.0000000  16.0000000
##        6  50.0000000  50.0000000
##        7  33.3333333  66.6666667
##        8  50.0000000  50.0000000
##        9   0.0000000 100.0000000
```

The **'1'** instructs the `prop.table()` function to calculate row proportions. You can also use the `plot()` function to represent this table graphically:

```
plot(prop.table(table(PARTNERS, HIV), 1) * 100, color = c("lightgreen", "red"))
```



The *chi-square test for trend* is an appropriate test to perform on this data. The `prop.trend.test()` function that performs the *chi-square test for trend* requires you to specify the *number of events* and the *number of trials*. In this table:

```
table(PARTNERS, HIV)
```

```
##          HIV
## PARTNERS   0   1
##        1  60   1
##        2 128   1
##        3 131   2
##        4  68   3
```

```
##         5  21   4
##         6   3   3
##         7   2   4
##         8   1   1
##         9   0   2
```

The *number of events* in each row is in the second column (labelled **1**) and the *number of trials* is the total number of cases in each row of the table.

We can extract this data from a table object:

```
tab <- table(PARTNERS, HIV)
events <- tab[ ,2]
trials <- tab[ ,1] + tab[ ,2]
```


```
tab <- table(PARTNERS, HIV)
events <- tab[ ,2]
trials <- tab[ ,1] + tab[ ,2]
```


Another way of creating the `trials` object would be to use the `apply()` function to sum the rows of the tab object:

```
trials <- apply(tab, 1, sum)
```

Pass this data to the `prop.trend.test()` function:

```
prop.trend.test(events, trials)
```

```
##
##  Chi-squared Test for Trend in Proportions
##
## data:  events out of trials ,
##  using scores: 1 2 3 4 5 6 7 8 9
## X-squared = 76.389, df = 1, p-value < 2.2e-16
```

With a linear trend such as this we can use `PARTNERS` in a logistic model without recoding or creating indicator variables. We can now specify and fit the logistic regression model:

```
gudhiv.lreg <- glm(formula = HIV ~ GUD + TRAVOUT + PARTNERS,
                   family = binomial(logit))
summary(gudhiv.lreg)
```

```
##
## Call:
## glm(formula = HIV ~ GUD + TRAVOUT + PARTNERS, family = binomial(logit))
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.70415  -0.19849  -0.11148  -0.06247   3.11742
```

```
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -9.4854     1.4663  -6.469 9.86e-11 ***
## GUD           1.3869     0.5937   2.336   0.0195 *
## TRAVOUT       2.0867     0.9547   2.186   0.0288 *
## PARTNERS      1.1605     0.2050   5.662 1.50e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 167.364  on 425  degrees of freedom
## Residual deviance:  99.377  on 422  degrees of freedom
##   (9 observations deleted due to missingness)
## AIC: 107.38
##
## Number of Fisher Scoring iterations: 8
```

We can use the `lreg.or()` function that we wrote earlier to calculate and display odds ratios and confidence intervals:

```
lreg.or(gudhiv.lreg)
```

```
##                OR  LCI   UCI
## (Intercept) 0.00 0.00  0.00
## GUD         4.00 1.25 12.81
## TRAVOUT     8.06 1.24 52.35
## PARTNERS    3.19 2.14  4.77
```

`PARTNERS` is incorporated into the logistic model as a continuous variable.

The odds ratio reported for `PARTNERS` is the odds ratio associated with a unit increase in the number of sexual `PARTNERS`. A man reporting five sexual partners, for example, was over three times as likely (odds ratio = 3.19) to have a positive HIV-2 serology than a man reporting four sexual partners.

An alternative approach would be to have created an *indicator* variables:

```
part.gt.5 <- ifelse(PARTNERS > 5, 1, 0)
```

This creates a new variable (`part.gt.5`) that indicates whether or not an individual subject reported having more than five sexual partners in the previous twelve months:

```
table(PARTNERS, part.gt.5)
```

```
##          part.gt.5
## PARTNERS   0   1
##        1  61   0
```

```
##         2 129   0
##         3 133   0
##         4  71   0
##         5  25   0
##         6   0   6
##         7   0   6
##         8   0   2
##         9   0   2
```

You can also inspect this on a case-by-case basis:

```
cbind(PARTNERS, part.gt.5)
```

```
##         PARTNERS part.gt.5
##    [1,]        2         0
##    [2,]        2         0
##    [3,]        1         0
##    [4,]        2         0
##    [5,]        3         0
##    [6,]        2         0
##    [7,]        4         0
##    [8,]        5         0
##    [9,]        2         0
##   [10,]        3         0
##   [11,]        4         0
##   [12,]        3         0
##   [13,]        1         0
##   [14,]        2         0
##   [15,]        5         0
##   [16,]        8         1
##   [17,]        5         0
##   [18,]        3         0
##   [19,]        2         0
##   [20,]        1         0
##   [21,]        2         0
##   [22,]        3         0
##   [23,]        2         0
##   [24,]        3         0
##   [25,]        4         0
##   [26,]        3         0
##   [27,]        4         0
##   [28,]        3         0
##   [29,]        4         0
##   [30,]        5         0
##   [31,]        4         0
##   [32,]        3         0
```

```
##  [33,]        4        0
##  [34,]        5        0
##  [35,]        2        0
##  [36,]        4        0
##  [37,]        3        0
##  [38,]        2        0
##  [39,]        1        0
##  [40,]        2        0
##  [41,]        3        0
##  [42,]        4        0
##  [43,]        3        0
##  [44,]        3        0
##  [45,]        2        0
##  [46,]        4        0
##  [47,]        5        0
##  [48,]        4        0
##  [49,]        3        0
##  [50,]        4        0
##  [51,]        1        0
##  [52,]        1        0
##  [53,]        2        0
##  [54,]        3        0
##  [55,]        3        0
##  [56,]        3        0
##  [57,]        3        0
##  [58,]        4        0
##  [59,]        5        0
##  [60,]        4        0
##  [61,]        3        0
##  [62,]        3        0
##  [63,]        5        0
##  [64,]        2        0
##  [65,]        2        0
##  [66,]        3        0
##  [67,]        2        0
##  [68,]        1        0
##  [69,]        2        0
##  [70,]        3        0
##  [71,]        2        0
##  [72,]        3        0
##  [73,]        4        0
##  [74,]        3        0
##  [75,]        3        0
##  [76,]        2        0
##  [77,]        5        0
```

```
##  [78,]        4         0
##  [79,]        3         0
##  [80,]        1         0
##  [81,]        2         0
##  [82,]        5         0
##  [83,]        3         0
##  [84,]        7         1
##  [85,]        6         1
##  [86,]        5         0
##  [87,]        5         0
##  [88,]        5         0
##  [89,]        4         0
##  [90,]        3         0
##  [91,]        2         0
##  [92,]        5         0
##  [93,]        1         0
##  [94,]        1         0
##  [95,]        1         0
##  [96,]        1         0
##  [97,]        2         0
##  [98,]        3         0
##  [99,]        4         0
## [100,]        3         0
## [101,]        3         0
## [102,]        2         0
## [103,]        3         0
## [104,]        4         0
## [105,]        3         0
## [106,]        2         0
## [107,]        3         0
## [108,]        4         0
## [109,]        3         0
## [110,]        3         0
## [111,]        4         0
## [112,]        3         0
## [113,]        6         1
## [114,]        3         0
## [115,]        4         0
## [116,]        3         0
## [117,]        3         0
## [118,]        2         0
## [119,]        3         0
## [120,]        4         0
## [121,]        7         1
## [122,]        3         0
```

```
## [123,]        2         0
## [124,]        3         0
## [125,]        4         0
## [126,]        3         0
## [127,]        2         0
## [128,]        3         0
## [129,]        4         0
## [130,]        8         1
## [131,]        5         0
## [132,]        6         1
## [133,]        5         0
## [134,]        4         0
## [135,]        4         0
## [136,]        4         0
## [137,]        3         0
## [138,]        4         0
## [139,]        3         0
## [140,]        3         0
## [141,]        2         0
## [142,]        2         0
## [143,]        1         0
## [144,]        2         0
## [145,]        1         0
## [146,]        2         0
## [147,]        3         0
## [148,]        1         0
## [149,]        2         0
## [150,]        3         0
## [151,]        2         0
## [152,]        2         0
## [153,]        2         0
## [154,]        1         0
## [155,]        1         0
## [156,]        2         0
## [157,]        3         0
## [158,]        3         0
## [159,]        2         0
## [160,]        3         0
## [161,]        4         0
## [162,]        2         0
## [163,]        5         0
## [164,]        4         0
## [165,]        2         0
## [166,]        3         0
## [167,]        2         0
```

```
## [168,]        2         0
## [169,]        1         0
## [170,]        4         0
## [171,]        3         0
## [172,]        3         0
## [173,]        2         0
## [174,]        3         0
## [175,]        2         0
## [176,]        4         0
## [177,]        3         0
## [178,]        2         0
## [179,]        3         0
## [180,]        4         0
## [181,]        2         0
## [182,]        3         0
## [183,]        3         0
## [184,]        4         0
## [185,]        2         0
## [186,]        3         0
## [187,]        2         0
## [188,]        2         0
## [189,]        3         0
## [190,]        3         0
## [191,]        2         0
## [192,]        3         0
## [193,]        2         0
## [194,]        4         0
## [195,]        3         0
## [196,]        2         0
## [197,]        2         0
## [198,]        3         0
## [199,]        2         0
## [200,]        3         0
## [201,]        2         0
## [202,]        3         0
## [203,]        3         0
## [204,]        2         0
## [205,]        3         0
## [206,]        2         0
## [207,]        3         0
## [208,]        2         0
## [209,]        1         0
## [210,]        6         1
## [211,]        9         1
## [212,]        1         0
```

```
## [213,]       2       0
## [214,]       3       0
## [215,]       4       0
## [216,]       5       0
## [217,]       4       0
## [218,]       5       0
## [219,]       5       0
## [220,]       5       0
## [221,]       4       0
## [222,]       3       0
## [223,]       4       0
## [224,]       3       0
## [225,]       2       0
## [226,]       1       0
## [227,]       2       0
## [228,]       3       0
## [229,]       2       0
## [230,]       1       0
## [231,]       4       0
## [232,]       3       0
## [233,]       4       0
## [234,]       3       0
## [235,]       3       0
## [236,]       2       0
## [237,]       2       0
## [238,]       1       0
## [239,]       2       0
## [240,]       3       0
## [241,]       2       0
## [242,]       1       0
## [243,]       2       0
## [244,]       4       0
## [245,]       3       0
## [246,]       2       0
## [247,]       3       0
## [248,]       2       0
## [249,]       2       0
## [250,]       1       0
## [251,]       2       0
## [252,]       3       0
## [253,]       2       0
## [254,]       3       0
## [255,]       1       0
## [256,]       2       0
## [257,]       3       0
```

```
## [258,]        2         0
## [259,]        4         0
## [260,]        3         0
## [261,]        3         0
## [262,]        2         0
## [263,]        2         0
## [264,]        1         0
## [265,]        1         0
## [266,]        1         0
## [267,]        1         0
## [268,]        1         0
## [269,]        1         0
## [270,]        1         0
## [271,]        1         0
## [272,]        1         0
## [273,]        1         0
## [274,]        2         0
## [275,]        3         0
## [276,]        2         0
## [277,]        3         0
## [278,]        2         0
## [279,]        1         0
## [280,]        2         0
## [281,]        3         0
## [282,]        4         0
## [283,]        3         0
## [284,]        2         0
## [285,]        3         0
## [286,]        2         0
## [287,]        1         0
## [288,]        2         0
## [289,]        4         0
## [290,]        7         1
## [291,]        1         0
## [292,]        4         0
## [293,]        1         0
## [294,]        3         0
## [295,]        3         0
## [296,]        4         0
## [297,]        3         0
## [298,]        2         0
## [299,]        2         0
## [300,]        1         0
## [301,]        2         0
## [302,]        3         0
```

```
## [303,]        3         0
## [304,]        3         0
## [305,]        2         0
## [306,]        4         0
## [307,]        4         0
## [308,]        5         0
## [309,]        4         0
## [310,]        4         0
## [311,]        9         1
## [312,]        3         0
## [313,]        3         0
## [314,]        2         0
## [315,]        2         0
## [316,]        1         0
## [317,]        1         0
## [318,]        2         0
## [319,]        7         1
## [320,]        3         0
## [321,]        2         0
## [322,]        1         0
## [323,]        2         0
## [324,]        4         0
## [325,]        6         1
## [326,]        5         0
## [327,]        3         0
## [328,]        2         0
## [329,]        3         0
## [330,]        4         0
## [331,]        3         0
## [332,]        2         0
## [333,]        3         0
## [334,]        4         0
## [335,]        3         0
## [336,]        3         0
## [337,]        2         0
## [338,]        3         0
## [339,]        4         0
## [340,]        3         0
## [341,]        2         0
## [342,]        3         0
## [343,]        1         0
## [344,]        1         0
## [345,]        2         0
## [346,]        3         0
## [347,]        4         0
```

```
## [348,]       3         0
## [349,]       3         0
## [350,]       2         0
## [351,]       4         0
## [352,]       5         0
## [353,]       4         0
## [354,]       3         0
## [355,]       3         0
## [356,]       2         0
## [357,]       2         0
## [358,]       1         0
## [359,]       4         0
## [360,]       1         0
## [361,]       1         0
## [362,]       4         0
## [363,]       3         0
## [364,]       2         0
## [365,]       1         0
## [366,]       4         0
## [367,]       1         0
## [368,]       2         0
## [369,]       3         0
## [370,]       1         0
## [371,]       5         0
## [372,]       4         0
## [373,]       3         0
## [374,]       2         0
## [375,]       1         0
## [376,]       2         0
## [377,]       3         0
## [378,]       2         0
## [379,]       4         0
## [380,]       2         0
## [381,]       3         0
## [382,]       4         0
## [383,]       7         1
## [384,]       3         0
## [385,]       2         0
## [386,]       4         0
## [387,]       4         0
## [388,]       3         0
## [389,]       2         0
## [390,]       2         0
## [391,]       1         0
## [392,]       2         0
```

```
## [393,]       6         1
## [394,]       7         1
## [395,]       2         0
## [396,]       1         0
## [397,]       2         0
## [398,]       3         0
## [399,]       1         0
## [400,]       2         0
## [401,]       3         0
## [402,]       2         0
## [403,]       1         0
## [404,]       2         0
## [405,]       3         0
## [406,]       2         0
## [407,]       3         0
## [408,]       2         0
## [409,]       3         0
## [410,]       2         0
## [411,]       4         0
## [412,]       2         0
## [413,]       2         0
## [414,]       1         0
## [415,]       2         0
## [416,]       3         0
## [417,]       2         0
## [418,]       3         0
## [419,]       2         0
## [420,]       3         0
## [421,]       2         0
## [422,]       3         0
## [423,]       4         0
## [424,]       2         0
## [425,]       2         0
## [426,]       3         0
## [427,]       4         0
## [428,]       4         0
## [429,]       1         0
## [430,]       2         0
## [431,]       3         0
## [432,]       2         0
## [433,]       1         0
## [434,]       1         0
## [435,]       2         0
```

We can now specify and fit the logistic regression model using our indicator variable:

```
gudhiv.lreg <- glm(formula = HIV ~ GUD + TRAVOUT + part.gt.5,
                   family = binomial(logit))
summary(gudhiv.lreg)
lreg.or(gudhiv.lreg)
```

```
##
## Call:
## glm(formula = HIV ~ GUD + TRAVOUT + part.gt.5, family = binomial(logit))
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6092  -0.2205  -0.2205  -0.0719   3.4521
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -5.9559     0.9850  -6.046 1.48e-09 ***
## GUD           1.4930     0.5805   2.572   0.0101 *
## TRAVOUT       2.2514     0.9319   2.416   0.0157 *
## part.gt.5     4.6791     0.7560   6.189 6.05e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 167.36  on 425  degrees of freedom
## Residual deviance: 106.43  on 422  degrees of freedom
##   (9 observations deleted due to missingness)
## AIC: 114.43
##
## Number of Fisher Scoring iterations: 7
```

```
##                OR   LCI    UCI
## (Intercept)  0.00  0.00   0.02
## GUD          4.45  1.43  13.89
## TRAVOUT      9.50  1.53  59.02
## part.gt.5  107.67 24.47 473.84
```

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).

5.1 Summary

- Using built-in functions and our own functions we can use `R` to analyse epidemiological data.
- The power of `R` is that it can be easily extended. Many user-contributed functions (usually packages of related functions) are available for download over the Internet. We will use one of these packages in the next exercise.

Chapter 6

Extending R with packages

R has no built-in functions for survival analysis but, because it is an extensible system, survival analysis is available as an add-in package. You can find a list of add-in packages at the R website.

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

Add-in packages are installed from the Internet. There are a series of R functions that enable you to download and install add-in packages.

The survival package adds functions to R that enable it to analyse survival data. This package may be downloaded and installed using `install.packages("survival")` or from the Packages or Packages & Data menu if you are using a GUI version of R.

Packages are loaded into R as they are needed using the `library()` function. Start R and load the survival package:

```
library(survival)
```

Before we go any further we should retrieve a dataset:

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

```
## The following objects are masked from ca (pos = 9):
##
##     group, status, time
```

The columns in this dataset on the survival of cancer patients in two different treatment groups are as follows:

time	Survival or censoring time (months)
status	Censoring status (1=dead, 0=censored)
group	Treatment group (1 / 2)

We next need to create a `survival` object from the `time` and `status` variables using the `Surv()` function:

```
response <- Surv(time, status)
```

We can then specify the model for the survival analysis. In this case we state that survival (`response`) is dependent upon the treatment `group`:

```
ca.surv <- survfit(response ~ group)
```

The `summary()` function applied to a `survfit` object lists the survival probabilities at each time point with 95% confidence intervals:

```
summary(ca.surv)
```

```
## Call: survfit(formula = response ~ group)
##
##                 group=1
##  time n.risk n.event survival std.err lower 95% CI upper 95% CI
##     8     22       1    0.955  0.0444        0.871        1.000
##     9     21       1    0.909  0.0613        0.797        1.000
##    13     19       1    0.861  0.0744        0.727        1.000
##    14     17       1    0.811  0.0856        0.659        0.997
##    18     16       1    0.760  0.0940        0.596        0.968
##    19     15       1    0.709  0.1005        0.537        0.936
##    21     14       1    0.659  0.1053        0.481        0.901
##    23     13       1    0.608  0.1087        0.428        0.863
##    30     10       1    0.547  0.1136        0.364        0.822
##    31      9       1    0.486  0.1161        0.305        0.776
##    32      8       1    0.426  0.1164        0.249        0.727
##    34      7       1    0.365  0.1146        0.197        0.675
##    48      5       1    0.292  0.1125        0.137        0.621
##    56      3       1    0.195  0.1092        0.065        0.585
##
##                 group=2
##  time n.risk n.event survival std.err lower 95% CI upper 95% CI
##     4     24       1   0.9583  0.0408      0.88163        1.000
##     5     23       2   0.8750  0.0675      0.75221        1.000
##     6     21       1   0.8333  0.0761      0.69681        0.997
##     7     20       1   0.7917  0.0829      0.64478        0.972
##     8     19       2   0.7083  0.0928      0.54795        0.916
##     9     17       1   0.6667  0.0962      0.50240        0.885
##    11     16       1   0.6250  0.0988      0.45845        0.852
##    12     15       1   0.5833  0.1006      0.41598        0.818
##    21     12       1   0.5347  0.1033      0.36614        0.781
##    23     11       1   0.4861  0.1047      0.31866        0.742
##    27     10       1   0.4375  0.1049      0.27340        0.700
```

```
##     28       9       1   0.3889  0.1039       0.23032         0.657
##     30       8       1   0.3403  0.1017       0.18945         0.611
##     32       7       1   0.2917  0.0981       0.15088         0.564
##     33       6       1   0.2431  0.0930       0.11481         0.515
##     37       5       1   0.1944  0.0862       0.08157         0.464
##     41       4       2   0.0972  0.0650       0.02624         0.360
##     43       2       1   0.0486  0.0473       0.00722         0.327
##     45       1       1   0.0000     NaN            NA            NA
```

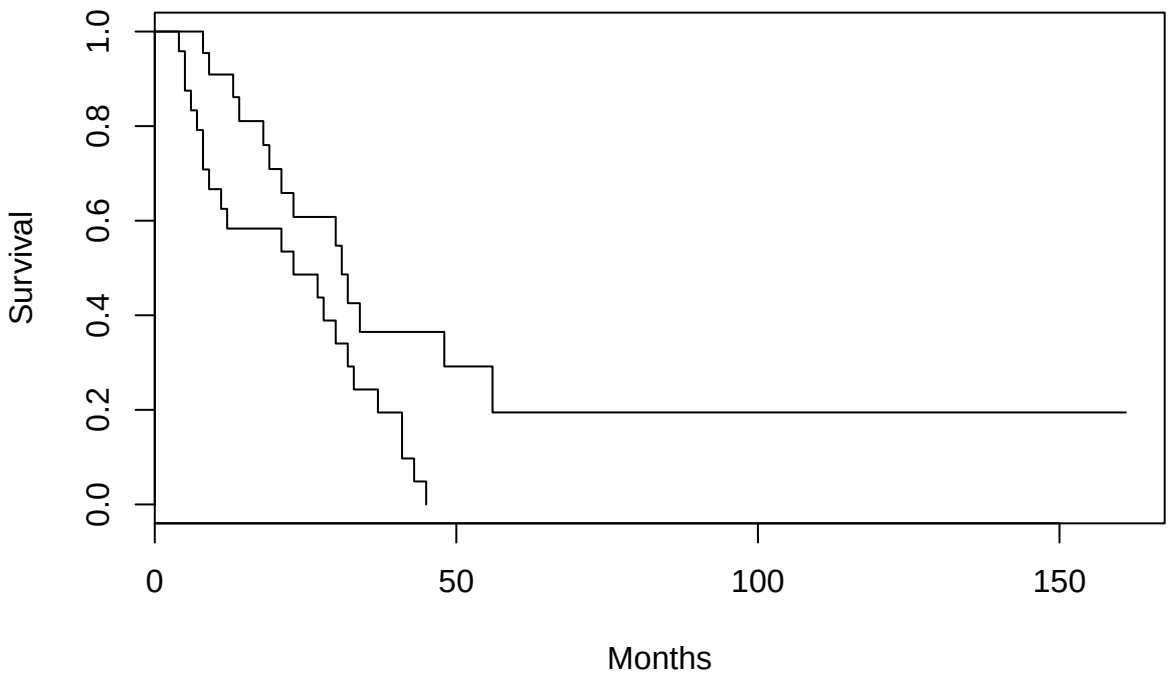
Printing the `ca.surv` object provides another view of the results:

```
ca.surv
```

```
## Call: survfit(formula = response ~ group)
##
##           n events median 0.95LCL 0.95UCL
## group=1 22     14     31      21      NA
## group=2 24     22     23      11      37
```

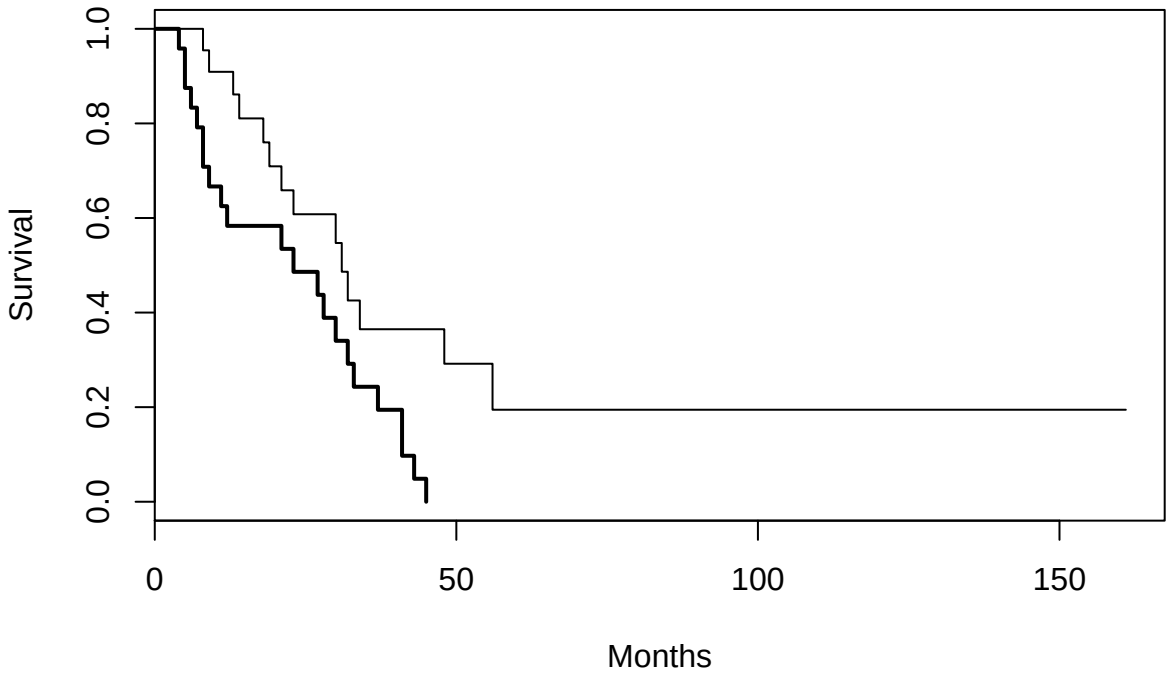
The `plot()` function with a `survfit` object displays the survival curves:

```
plot(ca.surv, xlab = "Months", ylab = "Survival")
```



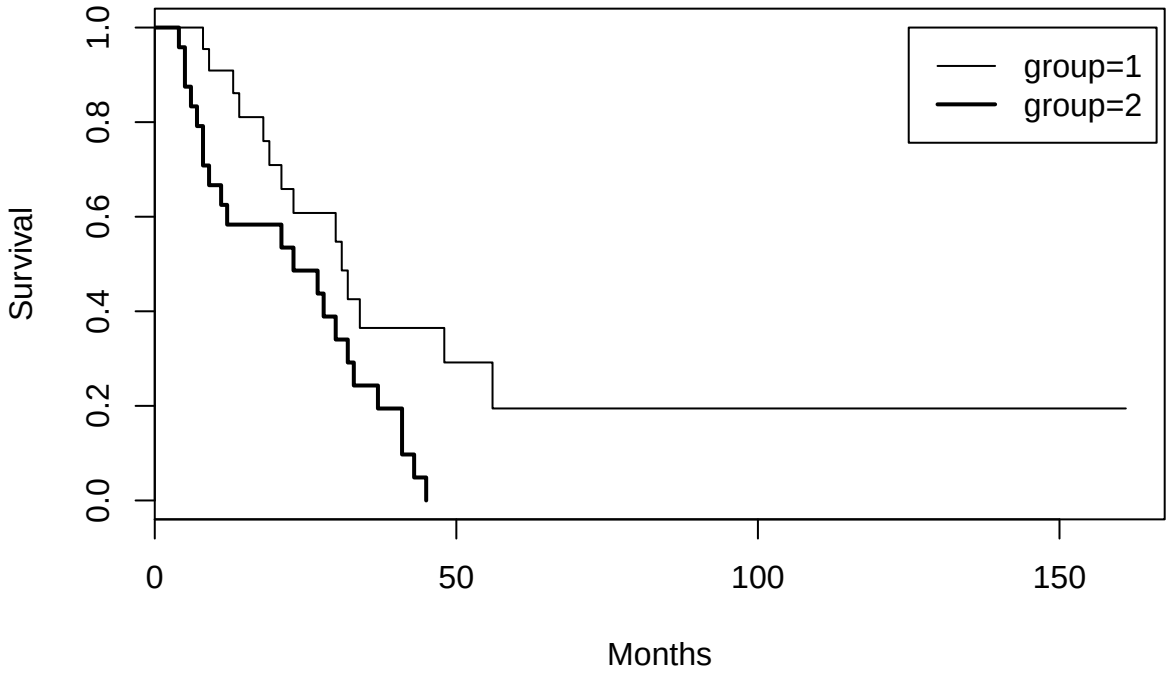
We can make it easier to distinguish between the two lines by specifying a width for each line using thelwd parameter of the `plot()` function:

```
plot(ca.surv, xlab = "Months", ylab = "Survival", lwd = c(1, 2))
```



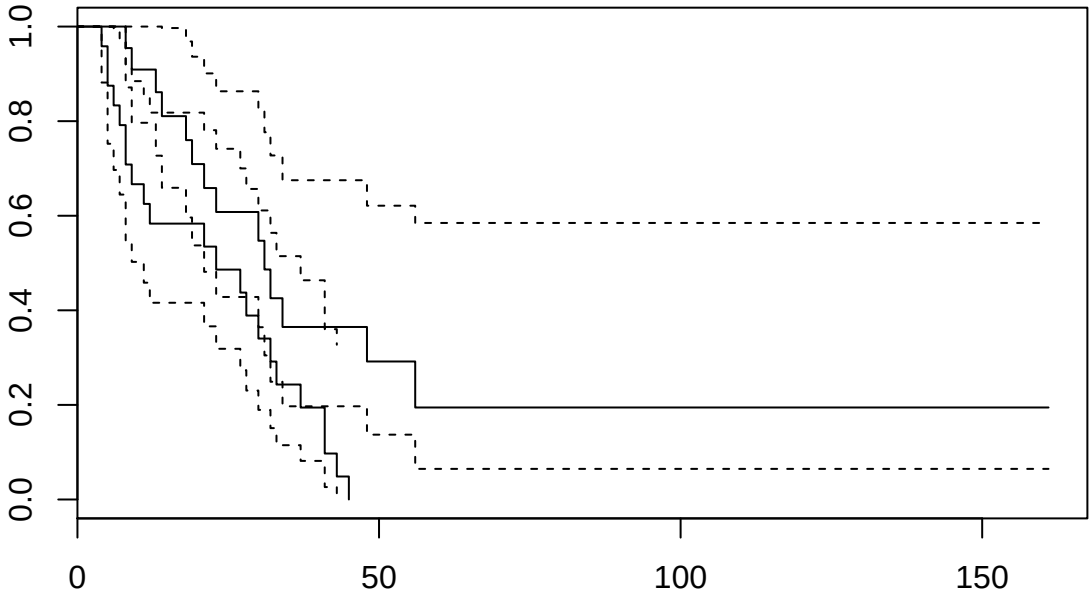
It would also be useful to add a legend:

```
legend(125, 1, names(ca.surv$strata), lwd = c(1, 2))
```



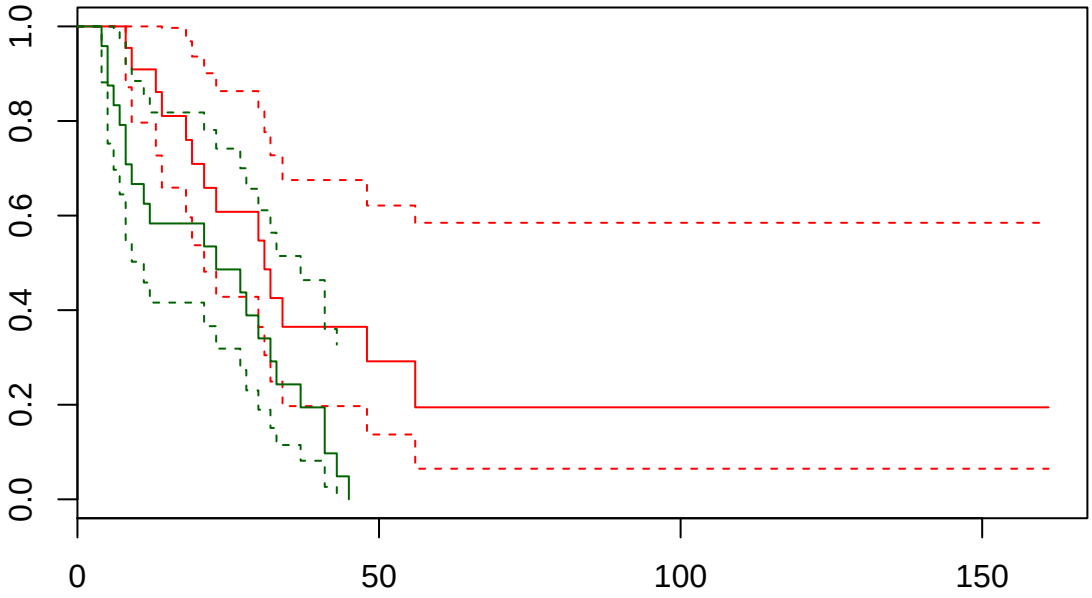
If there is only one survival curve to plot then plotting a `survfit` object will plot the survival curve with 95% confidence limits. You can specify that confidence limits should be plotted when there is more than one survival curve but the results can be disappointing:

```
plot(ca.surv, conf.int = TRUE)
```



Plots can be improved by specifying different colours for each curve:

```
plot(ca.surv, conf.int = TRUE, col = c("red", "darkgreen"))
```



We can perform a formal test of the two survival times using the `survdiff()` function:

```
survdiff(response ~ group)
```

```
## Call:
## survdiff(formula = response ~ group)
##
##          N Observed Expected (O-E)^2/E (O-E)^2/V
```

```
## group=1 22       14     21.1      2.38      6.26
## group=2 24       22     14.9      3.36      6.26
##
##  Chisq= 6.3  on 1 degrees of freedom, p= 0.0123
```

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).

6.1 Summary

- R can be extended by adding additional packages. Some packages are included with the standard R installation but many others are available and may be downloaded from the Internet.
- You can find a list of add-in packages at the R website: http://www.r-project.org/
- Packages may also be downloaded and installed from this site using the `install.packages()` function or from the **Packages** or **Packages & Data** menu if you are using a GUI version of R.
- Packages are loaded into R as they are needed using the `library()` function. You can use the `search()` function to display a list of loaded packages and attached data.frames.

Chapter 7

Making your own objects behave like R objects

In the previous exercises we concentrated on writing functions that take some input data, analyse it, and display the results of the analysis. The standard R functions we have used all do this. The `fisher.test()` function, for example, takes a `table` object (or the names of two variables) as input and calculates and displays the p- value for *Fisher's exact test* and the odds ratio and associated confidence interval for two-by-two tables:

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

```
## The following objects are masked from fem (pos = 6):
##
##     AGE, ANX, DEP, ID, IQ, LIFE, SEX, SLP, WT
```

```
## The following objects are masked from fem (pos = 8):
##
##     AGE, ANX, DEP, ID, IQ, LIFE, SEX, SLP, WT
```

```
## The following objects are masked from fem (pos = 9):
##
##     AGE, ANX, DEP, ID, IQ, LIFE, SEX, SLP, WT
```

```
## The following objects are masked from fem (pos = 14):
##
##     AGE, ANX, DEP, ID, IQ, LIFE, SEX, SLP, WT
```

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

```
##
##  Fisher's Exact Test for Count Data
##
## data:  SEX and 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
```

The results of the `fisher.test()` function may also be saved for later use:

```
ft <- fisher.test(SEX, LIFE)
ft
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  SEX and 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
```

The `fisher.test()` function returns an object of the class `htest`:

```
class(ft)
```

```
## [1] "htest"
```

which is a list containing the output of the `fisher.test()` function. Each item of output is stored as a different named item in the list:

```
names(ft)
str(ft)
```

```
## [1] "p.value"     "conf.int"    "estimate"    "null.value"  "alternative"
## [6] "method"      "data.name"
```

```
## List of 7
##  $ p.value    : num 0.0318
##  $ conf.int   : atomic [1:2] 1.08 14.21
##   ..- attr(*, "conf.level")= num 0.95
##  $ estimate   : Named num 3.62
##   ..- attr(*, "names")= chr "odds ratio"
##  $ null.value : Named num 1
##   ..- attr(*, "names")= chr "odds ratio"
##  $ alternative: chr "two.sided"
```

```
##  $ method     : chr "Fisher's Exact Test for Count Data"
##  $ data.name  : chr "SEX and LIFE"
##  - attr(*, "class")= chr "htest"
```

Each of these items can be referred to by name:

```
ft$estimate
ft$conf.int
```

```
## odds ratio
##   3.620646
```

```
## [1]  1.080298 14.214482
## attr(,"conf.level")
## [1] 0.95
```

When you display the output of the `fisher.test()` function either by calling the function directly:

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

```
##
##  Fisher's Exact Test for Count Data
##
## data:  SEX and 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
```

or by typing the name of an object created using the `fisher.test()` function:

```
ft
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  SEX and 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
```

The `print()` function takes over and formatted output is produced. The `print()` function

knows about `htest` class objects and produces output of the correct format for that class of object. This means that any function that produces an `htest` object (or any other standard `R` object) does not need to include `R` commands to produce formatted output.

All hypothesis testing functions supplied with `R` produce objects of the htest class and use the `print()` function to produce formatted output. For example:

```
tt <- t.test(WT ~ LIFE)
class(tt)
tt
```

```
## [1] "htest"
```

```
##
##  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
```

You can use this feature of `R` in your own functions. We will explore this by writing a function to test the null hypothesis that the *variance to mean ratio* of a vector of numbers is equal to one. Such a test might be used to investigate the spatial distribution (e.g. over natural sampling units such as households) of cases of a disease.

Create a new function using the `function()` function:

```
v2m.test <- function(data) {}
```

And start the function editor:

```
fix(v2m.test)
```

Now edit this function to make it do something useful:

```
function(data) {
  nsu <- length(data)
  obs <- sum(data)
  m <- obs / nsu
  v <- var(data)
  vmr <- v / m
  chi2 <- sum((data - m)^2) / m
  df <- nsu - 1
  p <- 1 - pchisq(chi2, df)
  names(chi2) <- "Chi-square"
```

```
  names(df) <- "df"
  names(vmr) <- "Variance : mean ratio"
  v2m <- list(method = "Variance to mean test",
              data.name = deparse(substitute(data)),
              statistic = chi2,
              parameter = df,
              p.value = p,
              estimate = vmr)
  class(v2m) <- "htest"
  return(v2m)
}
```

```
v2m.test <- function(data) {
  nsu <- length(data)
  obs <- sum(data)
  m <- obs / nsu
  v <- var(data)
  vmr <- v / m
  chi2 <- sum((data - m)^2) / m
  df <- nsu - 1
  p <- 1 - pchisq(chi2, df)
  names(chi2) <- "Chi-square"
  names(df) <- "df"
  names(vmr) <- "Variance : mean ratio"
  v2m <- list(method = "Variance to mean test",
              data.name = deparse(substitute(data)),
              statistic = chi2,
              parameter = df,
              p.value = p,
              estimate = vmr)
  class(v2m) <- "htest"
  return(v2m)
}
```

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

Before proceeding we should examine the `v2m.test()` function to make sure we understand what is happening:

1. The first eight lines after the opening curly bracket (`{`) contain the required calculations.

2. The next three lines use the `names()` function to give our variables names that will make sense in formatted output.

3. The next line creates a list of items that the function returns using some of the names

used by `htest` class objects.

4. The next line tells `R` that the list object called `v2m` is of the class `htest`.
5. The next line causes the function to return the `v2m` object (i.e. a list of class `htest` containing the named items `method`, `data.name`, `statistic`, `parameter`, `p.value`, and `estimate`).
6. The final line ends the function definition.

Note that objects of class htest may contain items with the following names:

Item	Usage
method	Text description of the test used to title output
data.name	Name(s) of data or variables used for the test
null.value	The null value
statistic	Value of test statistic
parameter	A test parameter such as the degrees of freedom of the test statistic
p.value	The p-value of the test
estimate	An estimate (e.g. the mean)
conf.int	Confidence interval of estimate
alternative	Text describing the alternative hypothesis
note	Text note

We are now ready to test the `v2m.test()` function. This table:

```
Number of cases :       0  1  2  3  4  6
Number of households : 24 29 26 14  5  2
```

shows the number of cases of chronic (stunting) undernutrition found in a random sample of 100 households.

We can reproduce the data behind this table using a combination of the `c()` and `rep()` functions:

```
stunt <- c(rep(0,24), rep(1,29), rep(2,26), rep(3,14), rep(4,5),
           rep(5,0), rep(6,2))
table(stunt)
```

```
## stunt
##  0  1  2  3  4  6
## 24 29 26 14  5  2
```

And use it to test our new `v2m.test()` function:

```
v2m.test(stunt)
```

Which should produce the following output:

```
v2m.test(stunt)
```

```
##
##  Variance to mean test
##
## data:  stunt
## Chi-square = 110.16, df = 99, p-value = 0.2083
## sample estimates:
## Variance : mean ratio
##                1.11274
```

If your `vm2.test()` function does not produce this output then use the `fix()` function:

```
fix(v2m.test)
```

to check and edit the `vm2.test()` function and try again.

The important thing to note from this exercise is that `R` allows us to specify a class for the output of our functions. This means that we can use standard `R` classes and functions to (e.g.) produce formatted output without us having to write commands to format the output ourselves.

More importantly, it also means that we can write functions that return values when we need them to return values but can also produce formatted output when we need them to produce formatted output.

Our `v2m.test()` function can produce values for later use:

```
vm <- v2m.test(stunt)
vm$p.value
```

```
## [1] 0.2083442
```

or produce formatted output:

```
v2m.test(stunt)
```

```
##
##  Variance to mean test
##
## data:  stunt
## Chi-square = 110.16, df = 99, p-value = 0.2083
## sample estimates:
## Variance : mean ratio
##                1.11274
```

This way of working is not limited to using standard `R` classes and functions.

`R` also allows us to define our own classes. We will explore this by defining functions and a new class to deal with two-by-two tables.

We need to create two functions:

1. One function will handle the calculations.
2. A second function function will produce formatted output when required.

Create a new function using the `function()` function:

```
rr22 <- function(exposure, outcome) {}
```

And start the function editor:

```
fix(rr22)
```

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))
  se.log.rr <- sqrt((b / a) / (a + b) + (d / c) / (c + d))
  lci <- exp(log(rr) - 1.96 * se.log.rr)
  uci <- exp(log(rr) + 1.96 * se.log.rr)
  rr22.output <- list(estimate = rr, ci = c(lci, uci))
  class(rr22.output) <- "rr22"
  return(rr22.output)
}
```

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

The `rr22()` function is similar to the `tab2by2()` function that you created in the second exercise of this tutorial except that the function now returns a list of values instead of formatted output:

```
fem <- read.table("fem.dat", header = TRUE)
attach(fem)
rr22.test <- rr22(SEX, LIFE)
names(rr22.test)
rr22.test$estimate
rr22.test$conf.int
rr22.test$conf.int[1]
rr22.test$conf.int[2]
```

```
## The following objects are masked from fem (pos = 3):
##
##     AGE, ANX, DEP, ID, IQ, LIFE, SEX, SLP, WT
```

```
## The following objects are masked from fem (pos = 7):
##
##     AGE, ANX, DEP, ID, IQ, LIFE, SEX, SLP, WT

## The following objects are masked from fem (pos = 9):
##
##     AGE, ANX, DEP, ID, IQ, LIFE, SEX, SLP, WT

## The following objects are masked from fem (pos = 10):
##
##     AGE, ANX, DEP, ID, IQ, LIFE, SEX, SLP, WT

## The following objects are masked from fem (pos = 15):
##
##     AGE, ANX, DEP, ID, IQ, LIFE, SEX, SLP, WT

## [1] "estimate" "ci"

## [1] 2.054167

## NULL

## NULL

## NULL
```

The function returns a list of class `rr22`:

```
class(rr22.test)
```

```
## [1] "rr22"
```

The displayed output from the `rr22()` function is, however, not pretty:

```
print(rr22.test)
rr22(SEX, LIFE)
```

```
## $estimate
## [1] 2.054167
##
## $ci
## [1] 0.966417 4.366232
##
## attr(,"class")
## [1] "rr22"

## $estimate
## [1] 2.054167
##
## $ci
## [1] 0.966417 4.366232
##
```

```
## attr(,"class")
## [1] "rr22"
```

This can be fixed by creating a new function:

```
print.rr22 <- function(x) {}
```

And start the function editor:

```
fix(print.rr22)
```

Now edit this function to make it do something useful:

```
function(x) {
  cat("RR     : ", x$estimate, "\n",
      "95% CI : ", x$ci[1], "; ", x$ci[2], "\n", sep = "")
}
```

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

The function name `print.rr22()` indicates that this function contains the print method for objects of class `rr22`. All objects of class `rr22` will use the function `print.rr22()` instead of the standard `R` `print()` function to produce formatted output:

```
rr22(SEX, LIFE)
rr22.test <- rr22(SEX, LIFE)
rr22.test
print(rr22.test)
```

```
## RR     : 2.054167
## 95% CI : 0.966417; 4.366232
```

```
## RR     : 2.054167
## 95% CI : 0.966417; 4.366232
```

```
## RR     : 2.054167
## 95% CI : 0.966417; 4.366232
```

Note that we can still extract returned values from an `rr22` class object:

```
rr22.test$estimate
```

The `print.rr22()` function only controls the way an entire `rr22` object is displayed.

You might like to use the `save()` function to save the `v2m.test()`, `rr22()`, and `print.rr22()` functions before quitting `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).

7.1 Summary

- `R` objects can be assigned a class or type.
- Objects of a specific class or type may share functions that extract and manipulate data common to members of that class. This allows you to write functions that handle data that is common to all members of that class (e.g. to produce formatted output for hypothesis testing functions).
- `R` provides a set of ready-made classes (e.g. `htest`) which can be used by standard R functions such as the `print()` and `summary()` functions.
- `R` allows you to create new classes and class-specific functions that can extract and manipulate data common to the new classes.
- Classes allows you to create versatile functions that return values when you need them to return values but can also produce formatted output when you need them to produce formatted output.
- Classes allow you to write functions that can be chained together so that the output of one function is the input of another function.

Chapter 8

Writing your own graphical functions

R provides a pretty full set of graphical functions for plotting data as well as `plot()` methods for a wide variety of statistical functions. There will be times, however, when you will need to write you own graphical functions to present and analyse data in a specific way. In this exercise we will create a function that produces a plot that may be used for assessing agreement between two methods of clinical measurement as described in:

`Bland JM, Altman DG. Statistical Methods for Assessing Agreement Between Two Methods of 310.`

Which involves plotting the difference of two measurements against the mean of the two measurements and calculating and displaying limits of agreement.

Start R and retrieve and attach the sample dataset:

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

```
## The following objects are masked from ba (pos = 9):
##
##     Mini, Wright
```

The `ba` data.frame contains measurements (in litres per minute) taken with a *Wright peak flow meter* and a *Mini-Wright peak flow meter*. This is the same data that is presented in the referenced Lancet article:

```
##    Wright Mini
## 1     494  512
## 2     395  430
## 3     516  520
## 4     434  428
## 5     476  500
## 6     557  600
## 7     413  364
## 8     442  380
```

```
## 9     650  658
## 10    433  445
## 11    417  432
## 12    656  626
## 13    267  260
## 14    478  477
## 15    178  259
## 16    423  350
## 17    427  451
```

Wright Mini 1 494 512 2 395 430 3 516 520 4 434 428 5 476 500 6 557 600 7 413 364 8 442 380 9 650 658 10 433 445 11 417 432 12 656 626 13 267 260 14 478 477 15 178 259 16 423 350 17 427 451

You can examine the `ba` data.frame using the `print()` and `summary()` functions:

```
print(ba)
ba
summary(ba)
```

```
##    Wright Mini
## 1     494  512
## 2     395  430
## 3     516  520
## 4     434  428
## 5     476  500
## 6     557  600
## 7     413  364
## 8     442  380
## 9     650  658
## 10    433  445
## 11    417  432
## 12    656  626
## 13    267  260
## 14    478  477
## 15    178  259
## 16    423  350
## 17    427  451
```

```
##    Wright Mini
## 1     494  512
## 2     395  430
## 3     516  520
## 4     434  428
## 5     476  500
## 6     557  600
## 7     413  364
```

```
## 8     442  380
## 9     650  658
## 10    433  445
## 11    417  432
## 12    656  626
## 13    267  260
## 14    478  477
## 15    178  259
## 16    423  350
## 17    427  451
```

```
##      Wright           Mini
##  Min.   :178.0   Min.   :259.0
##  1st Qu.:417.0   1st Qu.:380.0
##  Median :434.0   Median :445.0
##  Mean   :450.4   Mean   :452.5
##  3rd Qu.:494.0   3rd Qu.:512.0
##  Max.   :656.0   Max.   :658.0
```

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

```
ba.plot <- function(a, b) {}
```

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

```
fix(ba.plot)
```

We will start be writing a basic function which we will gradually improve throughout this exercise.

Chapter 9

More graphical functions

Chapter 10

Computer intensive methods

Chapter 11

What now?