Utts and Heckard's

Mind on Statistics

Asia-Pacific Edition

Technical Manual

A. Jonathan R. Godfrey [1]

ISBN: 9780170183024

[1]All comments or suggestions regarding this document should be sent to Jonathan Godfrey by emailing <a.j.godfrey@massey.ac.nz>.

Contents

Preface

This manual supports the Asia-Pacific edition of Utts and Heckard's *Mind on Statistics*. It uses the software known as R for as many examples from the text as possible. Many figures are also reconstructed to illustrate the differnces between R and the software used by the authors to generate the figures. Whenever the data for a specific example is not available to students, we have chosen to present the technique using a different dataset from the book or made available with the base installation of R.

Versions of R come and go. This manual can be easily updated for different versions of R, but the text you are reading was created using version 2.15.2 which was released on 26 October, 2012. All code should work on other versions equally well. Please report any faults with the code to the author.

Jonathan Godfrey <a.j.godfrey@massey.ac.nz>
October 31, 2012

Chapter 0

Getting started with R and *Mind on Statistics*

Note: The structure of this chapter and some of its sections are heavily influenced by the R Technical Manual that accompanies the U.S. edition of the text. The author's thanks are gratefully extended to its authors and the publishers for copyright permission.

0.1 Manual overview

The purpose of this manual is to help you learn how to use the free R software to create the graphs, reproduce the simulations, and perform the data analyses presented by *Mind on Statistics*. This chapter will describe how to obtain R and give a brief introduction to the software. The following chapters will provide specific commands for the examples in the book.

Line-by-line examples of R code will be provided for many of the examples demonstrated in the text. Users of R are working in many different ways. The first observation is that R is available to users of all major operating systems, and the code we all use is the same. If you choose to use the version of R with pull down menus, you will still need to use the code given in this manual somehow. You will either type it into R line by line at the prompt, or use the script window within R to have a set of commands all editable at the same time. There are no screen shots of R in this manual. This is done to future-proof the manual so that users down the track are not confused by a different appearance and the consequential musings about 'what have I done wrong?' which are real for some but an unnecessary distraction for all.

All chapter headings from Chapter 1 to 14 match those given in the Asia-Pacific edition of *Mind On Statistics*; this extends to all sections, tables and figure references as well. Page numbers refer to the textbook unless explicit mention of this manual follows the reference. An index is given which lists many terms that are not explicitly mentioned in the Table of Contents as well as most of the R commands used. This should help work backwards from R code so that the reader can put a command into context. The pdf version of this manual has hyper-links and bookmarks to assist navigation, including links between the index and the text.

We assume that this manual is read after the relevant section of the main text. Full explanations of why a particular technique is used are not given in this manual because we expect the reader has purchased the main text and has this resource only to assist in using R.

0.2 What is R?

R is a computer language and environment that was developed with statistical graphics and analysis in mind.

Consequently it is commonly thought of as a statistical software package, like the proprietary Minitab and SPSS packages. In the growing atmosphere of free software, scientists are constantly making available new packages that enable R to perform very advanced modern statistics. This manual, however, will focus on the more elementary aspects of R needed to learn statistical concepts and successfully perform the statistics that you are likely to encounter in your future careers.

There are several consequences of R being free software developed by scientists for scientists. First of all, it is very powerful. If you decide to continue in a career that depends heavily on statistics such as economics, biology, medicine, marketing, etc., R will allow you to develop your own statistical functions specific to your own immediate needs. Secondly, it was created as a tool for scientists rather than for mass marketing to make money. Thus it is line command driven and lacks features as pull down menus and point-and-click commands. This results in software that has a high 'nerd factor' as you will notice when looking at the help commands and manuals. S-Plus, a proprietary software package, is almost identical to R with respect to line commands, but includes pull-down menus and some point-and-click commands. There are some pull-down menus in R but these are for the management of the software, not for performing analyses.

There are a number of projects aimed at providing R users with a graphical user interface for performing statistical analyses. None of these has met with uniform support to date, although some are held in high regard — all have their critics too! We do not use any of these 'front-ends' because their development is fast-paced and we cannot provide a manual that keeps up with the changes. All these front-ends rely on the code that appears in this manual to do the actual work. Each front-end has its own manual so the interested user can investigate the advantages of using a front-end if they choose.

The sharing of work among R users is extensive. When users have developed a set of functions and datasets that are worthy of sharing, they are made available as a package. Add-on packages number into the thousands and we certainly do not need to use time looking through them until we find the functionality that is not part of the base distribution of R.

0.3 Obtaining R

R is freely available via the website:

http://www.r-project.org

The first task to achieve when wanting to get the installation file for R is to choose a 'CRAN mirror' which simply means to choose where we wish to obtain the file we need. There are lots of options for servers around the world and you need to select one (probably close to you) now.

We now choose the operating system we are using. R is available for many platforms; choose the most appropriate for you. Then look to download the 'Base distribution' of R. The vast majority of users do not need source code or any additional packages at first.

If you are on the right track, you'll now be downloading a file to your own PC. If you are a Windows user, this is a file with a name something like `R-2.12.0-win.exe` — the name includes the version number which is bound to change soon!

Once this file is on your own PC, you can start the installation by clicking on the file. We recommend using the default options during the installation.

0.4 Basic commands

R can be used as a calculator, for example:

```
> 1 + 1
[1] 2
> 2 * 5
[1] 10
> 10/3
[1] 3.333
> 2^3
[1] 8
```

The lines of code that start with a greater than symbol > are typed in exactly as shown, and executed once the 'Enter' key is hit. Extra spaces do not matter. If the command is incomplete for some reason, the prompt will change from > to + to suggest you should add some more detail.

These are the most basic things we can do in R. Almost everything else we will do is done using commands entered at the prompt. A command needs to work on something, called an argument. For example:

```
> sqrt(25)
[1] 5
```

is the command for getting the square root of 25; in this instance the argument supplied to the `sqrt()` command is `25`. Sometimes arguments are variable names, sometimes necessary details for the command, and sometimes even other commands. The parentheses are necessary and must be paired.

We can assign the outcome from any calculation using the equals sign after a name. For example:

```
> x = 25
> y = 1000
```

which has the benefit of then being able to be used as the argument for another command:

```
> sqrt(x)
[1] 5
```

Note that if we do assign a value to a name, R does not print the result for us. If we want to know anything about the value(s) stored in a named object, we just type its name:

```
> y
[1] 1000
```

One of the greatest advantages R has is the ability to edit a previously issued command. Hitting the up and down arrow keys moves us through the history of issued commands. Try this now.

Other points to note are:

- The hash character # tells R to ignore everything else until the Enter key is hit. This is so we can add comments to our work.
- R is case sensitive. `mean` is not the same as `Mean`, nor `MEAN`.
- To see what objects have been created, use the `ls()` command. For this chapter we have:

  ```
  > ls()
  [1] "x" "y"
  ```

- We can remove a named object from our workspace by using the `rm()` command.

  ```
  > rm(x)
  > ls()
  [1] "y"
  ```

- We can get out of our R session by typing `q()` at the prompt. We will be asked if we want to save our session.

0.5 Storing data: Objects, vectors, matrices, and data frames

In the previous section, we showed how to store a single number to a named object. R can use a name for objects of many kinds. A vector is just a single list of values all belonging together as a single set. This might best be thought of as a variable because this is the most common type of vector we will encounter. Two examples of how to enter a vector of values follow:

```
> Days = 1:30
> Days
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
[24] 24 25 26 27 28 29 30
> Months = c("April", "June", "September", "November")
> Months
[1] "April"     "June"      "September" "November"
```

The first example is a set of integers from 1 to 30. The second is a set of test strings, or character values, which were entered using the `c()` command. Note how R printed out the contents of the named object when we typed in the name. Take special note of the numbers in square brackets. The first will always be a `[1]` because the next item listed is the first. If a second row of elements is printed out because the set did not appear on a single line, then the next number in square brackets tells you where in the list we are up to. This is quite useful when we move to less trivial datasets.

A matrix is similar to a vector except that it has two dimensions, not one. A data frame is different in that while it is two-dimensional, the contents are not all of the same type. Usually, a data frame is made up of a collection of vectors which might differ in type — some numeric, some textual. Each time we use a dataset throughout this manual we are working with a data frame. We may create matrices for some analyses but this is the exception, not the rule.

0.6 Importing data into R

When data are transferred from person to person, we will not always get it in the desired format. Whenever we want to import data into R we need to know how it is formatted. The best options are text files or what are known as 'comma separated values' files.

Text files come in many forms, with extensions like `txt`, `dat`, etc. and in fact, even the `csv` files we prefer are just text files too. The differences come from how the data are presented in these files. A `csv` file has a single line for each row of the spreadsheet it came from with all columns being separated by commas. Other text files use spaces or tabs to separate the data and this often leads to confusion, especially if a cell in the original spreadsheet is actually blank.

We suggest:

1. Open your spreadsheet application and enter some data. Perhaps you have a dataset on hand that could be used here.

2. Save this data as a 'comma separated value' file.

3. Move this `csv` file to your working directory in R. If you do not know where this is, type `getwd()` at the command prompt in R.

4. Import the data into R using the `read.csv()` command. If your file is called `MyFirstData.csv` then the command will be:

   ```
   > MyData = read.csv("MyFirstData.csv")
   ```

5. See if it has been successfully imported by issuing a few commands: First is the dataset the right size?

   ```
   > dim(MyData)
   ```

   Second, did the top row of the file get treated as we expected?

   ```
   > str(MyData)
   ```

6. If it all worked then the default settings of the `read.csv()` command were suitable for this dataset. If not, then we need to tell R that you did or did not have a header row of variable names using:

   ```
   > MyData = read.csv("MyFirstData.csv", header = TRUE)
   ```

   or,

   ```
   > MyData = read.csv("MyFirstData.csv", header = FALSE)
   ```

If successful, you now have a data frame stored in the current workspace with the name `MyData`. Well done.

0.7 Importing data from the *Mind on Statistics* CD into R

Importing data is one of the greatest headaches for users of all statistical software. Even using cut-and-paste techniques can prove troublesome as the data may not transfer in the expected fashion. We experienced enough grief working with the data supplied on the CD that accompanies Mind on Statistics that we decided to help you avoid having to check the data over and over again. The *MindOnStats* package is the result of importing the datasets from a usable format into R and then saving them as data files within R. Help pages for the data were created (fairly sparse ones, actually) and the package created. All you need to do is to obtain the package file from the same source as this manual and install it when you open R. To get to the data within the package we will use the `data()` command and this is done in each chapter a dataset is used.

0.8 Installing the necessary *MindOnStats* package

This manual assumes you have installed the add-on package we have developed called *MindOnStats*. If you are a Windows user then you will find the package in a file called something like `MindOnStats-0.4.zip` but note the last few numbers will change as the package is updated. Users of other operating systems will use the file `MindOnStats-0.4.tar.gz` instead. These files should be found in the same location where you obtained this manual.

Download the appropriate file to your local machine. Then open R and go to the 'Packages' menu. The menu item you want is the last one, 'Install package(s) from local zip file...' — well, that's where it is for Windows users.

You should now be able to direct R to the zip file you saved on your machine. You will see confirmation messages once R has finished with the installation. You are now ready to use the datasets that accompany both the Asia-Pacific and United States editions of *Mind on Statistics* by Utts and Heckard.

0.9 Editing data

Editing data within R is possible but best avoided. The `edit()` command will allow you to edit the data concerned but the results must be assigned to a named object if they are to be saved. Some experimentation might be required.

We recommend using the best software to do every job. A spreadsheet application is probably a better option for entering data than using the `edit()` command. Having entered data into a spreadsheet, you should then see the section on Importing data into R.

0.10 Exporting data

The complementary command for `read.csv()` is `write.csv()`. This command needs to know two things: First, what is the R data frame you want to export, and second, which file do you want it stored in? This task is not required in this manual so we leave it to the interested reader to investigate the help for the `write.csv()` command with the confidence that this is an easily achieved task.

0.11 Getting help

If you need help with the syntax of particular commands, such as the `mean()` command, you might type `?mean` or whatever command is of interest. It is probably a much better idea to emulate the code given in an example from the text. That's why this manual exists.

If you are not quite sure of the name for a particular task but have a key word in mind, you can type `??bartlett` — a person's name in this instance. This gives you a list of the help pages that have this word in them.

Other useful documents for R do exist. Unfortunately, some less than useful ones exist as well. Students will need to be careful when choosing suitable resources. At this stage, it is probably a good idea that students stick to the resources made available to them by the staff responsible for their course.

0.12 Learning more about R

You will continue to learn more about R as you progress through this manual, which replicates the examples from each chapter in *Mind on Statistics*. Although this manual will focus on more introductory statistics, the potential for using R for statistical analysis is almost endless. There always seems to be more about this software and statistics that a person can learn, no matter how introductory or advanced the user. Besides the R manuals available through the Help icon at the top of R, there are a number of books written at introductory and advanced levels which describe how to use R and the similar S-Plus package.

Chapter 1

Thinking statistically

There is no work requiring R in this chapter, but see the example in Section 1.3 below.

1.1 What is statistics?

There is no work requiring R in this section.

1.2 Some statistical stories in real and complex problems

There is no work requiring R in this section.

1.3 Some examples of what can go wrong at the beginning of data investigations

There is no work requiring R in this section, but it is timely to show some useful commands that can help identify problems in the way data are recorded.

The `str()` command presents the structure of a data set. The `head()` command prints the first six rows of the data set. It is quite useful to see the way a data set is arranged before the user gets too far into creating graphs and numerical summaries of the data, let alone any analysis.

The default installation of R includes a number of data sets provided for testing purposes and are directly available to the user. Try:

```
> head(quakes)
     lat  long depth mag stations
1 -20.42 181.6   562 4.8       41
2 -20.62 181.0   650 4.2       15
```

```
3 -26.00 184.1    42 5.4       43
4 -17.97 181.7   626 4.1       19
5 -20.42 182.0   649 4.0       11
6 -19.68 184.3   195 4.0       12
> str(quakes)
'data.frame':        1000 obs. of  5 variables:
 $ lat     : num  -20.4 -20.6 -26 -18 -20.4 ...
 $ long    : num  182 181 184 182 182 ...
 $ depth   : int  562 650 42 626 649 195 82 194 211 622 ...
 $ mag     : num  4.8 4.2 5.4 4.1 4 4 4.8 4.4 4.7 4.3 ...
 $ stations: int  41 15 43 19 11 12 43 15 35 19 ...
```

and if you haven't guessed what the data is for, look at its help page by typing `?quakes` at the command prompt. This will bring up the help page for this data set.

1.4 The discovery of knowledge

There is no work requiring R in this section.

Chapter 2

Gathering and preparing useful data

There are few direct needs for R in this chapter, but showing how R is used to replicate certain displays needs to be dealt with somewhere.

You need to ensure the *MindOnStats* package is installed and therefore available for use in the current session. This gives direct access to the relevant datasets. You can do this by issuing the `library()` command:

```
> library(MindOnStats)
```

Be careful to use the correct case for the name of the package. This command is not actually necessary if we are just interested in obtaining data from a package, but it is a good habit to think about the packages we need in an R session. It is easy to install any additional packages as they are required; see the appendix for guidance.

2.1 Data sets and types of investigations

The data in Example 2.1 can be obtained using the `data()` command:

```
> data(Bike, package = "MindOnStats")
```

This is a good opportunity to show students why we do not just want to print out the entire dataset without thinking about it first. This dataset is quite large and has a number of variables. We should see how R handles the various data types in particular using the `str()` command:

```
> str(Bike)
'data.frame':        945 obs. of  5 variables:
 $ Time     : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Type     : Factor w/ 4 levels "Bike","Jog","Other",..: 1 4 1 1 1 1 1 2 1 1 ...
 $ Speed    : int  30 5 36 26 21 28 37 10 27 34 ...
 $ Direction: Factor w/ 2 levels "In","Out": 1 2 1 1 1 1 1 1 1 1 ...
 $ Gender   : Factor w/ 2 levels "Female","Male": 2 1 2 2 2 2 2 1 2 2 ...
```

and perhaps the `head()` command:

```
> head(Bike)
  Time Type Speed Direction Gender
1    1 Bike    30        In   Male
2    1 Walk     5       Out Female
3    1 Bike    36        In   Male
4    1 Bike    26        In   Male
5    1 Bike    21        In   Male
6    1 Bike    28        In   Male
```

A shorter summary of the size of the dataset can be obtained using the `dim()` command:

```
> dim(Bike)
[1] 945   5
```

2.2 Some practicalities and challenges in planning data investigations

No examples in this section require use of R.

2.3 Types of data and variables

Use of the `str()` command shows the user what data type R thinks is appropriate for each variable in a dataset. Note how variables that take integer values are given the classifier `int` for integer, while purely continuous variables are given the classifier `num` for numeric. Categorical variables, and sometimes discrete valued variables, are called *factors* by R. This has the advantage of forcing certain styles of presentation to be the default action without needing to change the command — it's already been taken care of when the programming was done.

2.4 Surveys

We see the calculation of the margin of error here. Use of R as a basic calculator is possible and requires knowledge of the specific commands required. It's more obvious that the `sqrt()` command gives us the square root, but sometimes the reverse is not quite so obvious.

```
> 1/sqrt(1600)
[1] 0.025
```

This calculation uses the abbreviation common to many programs (the `sqrt()` command) for the square root, but note the division sign and the fact that we do not need to enter an equals sign to get the answer. Also note the `[1]` printed to the left of the answer. This shows the user that the following item is the first. This feature is more use when the answer is a larger group of items which are printed on multiple lines in the output.

To replace the Minitab tip given on page 31 for selecting a random sample of ten students from a class of 270, do the following:

```
> sample(270, size = 10)
 [1]  80 209 145  57 151  63  91 196 139 134
```

Note that the `sample()` command has *sampling without replacement* as its default behaviour so this argument does not need to be stipulated when we enter this command. Many R commands have default settings that do not need to be entered to have an effect.

2.5 Designing experiments

No examples in this section require use of R.

2.6 Some types of observational studies

No examples in this section require use of R.

2.7 Some cautions in experiments and observational studies

No examples in this section require use of R.

Chapter 3

Turning data into graphical information

Most statistical software has default settings for creating graphics. In an introductory course where understanding is more important than presentation it is preferable to use time getting the right information rather than getting the most beautiful and totally perfect graph for publication. With this in mind, the examples given for this chapter use the default actions of R, unless these result in misleading the user of the graph; in such circumstances, the bad graph will be replaced by an alternative that relies on adding greater detail to the code used to generate the graph. In some circumstances we just add detail to the commands; experiment by removing arguments from the commands at your own discretion.

3.1 Categorical data

Example 3.1 uses the `Bike` dataset. Obtain it using:

```
> data(Bike, package = "MindOnStats")
```

The `summary()` command gives some useful information on all variables. We can choose to get information on only one variable at a time by using the $ notation.

```
> summary(Bike)
      Time            Type          Speed        Direction    Gender   
 Min.   : 1.0   Bike :625   Min.   : 3.0   In :438   Female:253  
 1st Qu.:12.0   Jog  :110   1st Qu.:10.0   Out:507   Male  :692  
 Median :24.0   Other:  6   Median :23.0                         
 Mean   :23.3   Walk :204   Mean   :19.7                         
 3rd Qu.:36.0               3rd Qu.:27.0                         
 Max.   :40.0               Max.   :41.0                         
> summary(Bike$Type)
 Bike   Jog Other  Walk 
  625   110     6   204 
```

We should note the different behaviour of the `summary()` command for the different data types here. R users benefit from this flexibility as it is built into several commonly used commands (you'll see more of this later).

To generate the information in Table 3.1 we will use the following series of commands:

```
> tapply(Bike$Time, Bike$Type, length)
 Bike   Jog Other  Walk
  625   110     6   204
> tapply(Bike$Time, Bike$Type, length)/length(Bike$Type)
    Bike      Jog    Other     Walk
0.661376 0.116402 0.006349 0.215873
```

Note that the `tapply()` command provides an alternative to the `summary()` command in this instance. It needs something to do work on to start with, given as the first argument. The second argument is for the groups to be identified, which is actually what we are most interested in in this example. The third argument is the actual function we are applying. The `length()` command counts up how many items there are. This command is then used again to convert the frequencies into relative frequencies in the second step.

The default *pie chart* and *bar chart* are given in the first version of Figure 3.1 on page 16 of this manual. The pie chart does not include the relative frequency information without some extra juggling of information. See the second version of Figure 3.1 on page 17 of this manual.

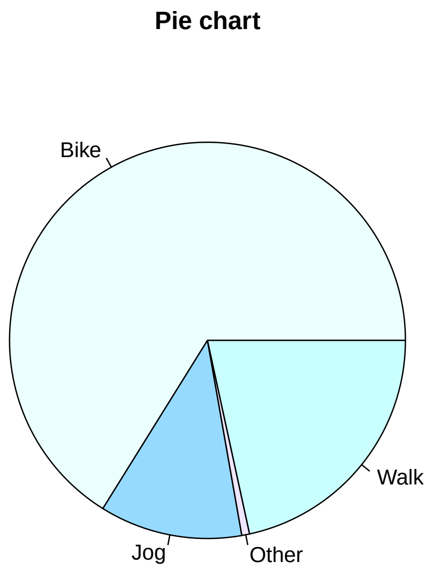
Note that without the division of the summary values by the total number of responses, the bar chart would present the counts rather than the relative frequencies. The enhanced graphs are given on page 17 of this manual.

To generate Table 3.2 on page 62, we use the `tapply()` command again, this time using the `list()` command to add a two-way classification as the second argument:

```
> tapply(Bike$Time, list(Bike$Gender, Bike$Type), length)
       Bike Jog Other Walk
Female  117  30     2  104
Male    508  80     4  100
> t(tapply(Bike$Time, list(Bike$Gender, Bike$Type), length))
      Female Male
Bike     117  508
Jog       30   80
Other      2    4
Walk     104  100
> addmargins(t(tapply(Bike$Time, list(Bike$Gender, Bike$Type),
       length)))
      Female Male Sum
Bike     117  508 625
Jog       30   80 110
Other      2    4   6
Walk     104  100 204
Sum      253  692 945
```

Figure 3.1 (using the simplest code) on page 61 of Utts and Heckard.

```
> pie(summary(Bike$Type))
> title("Pie chart")
```



```
> barplot(summary(Bike$Type))
> title("Bar chart", ylab = "Count", xlab = "Type")
```

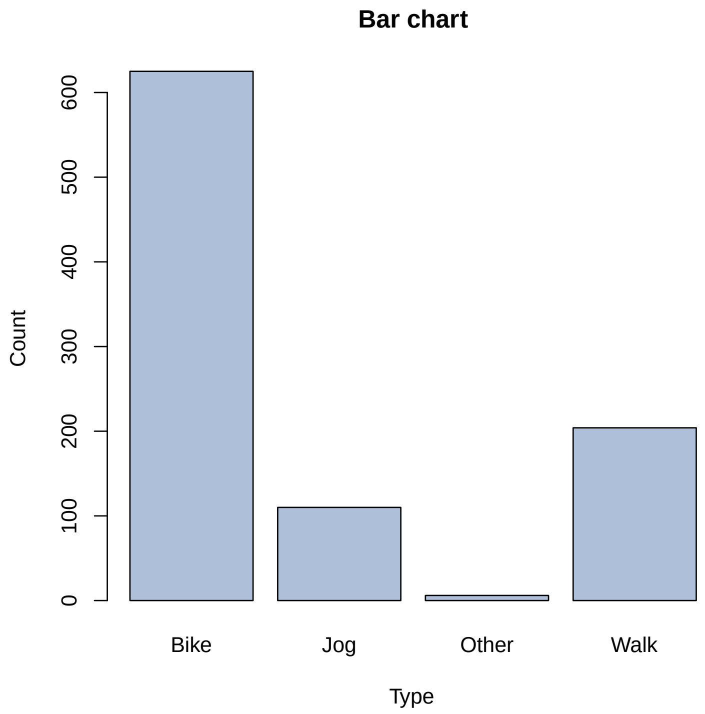
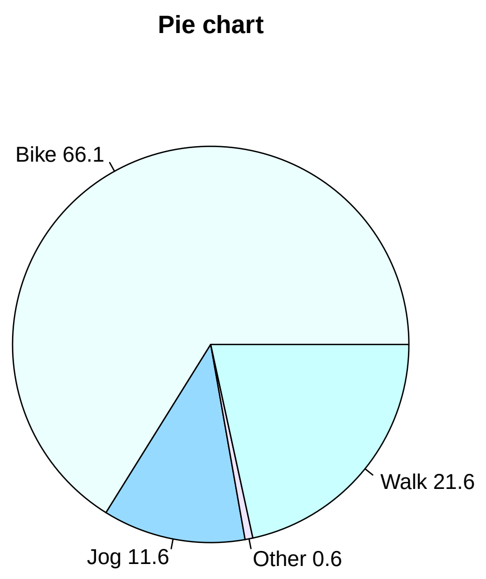
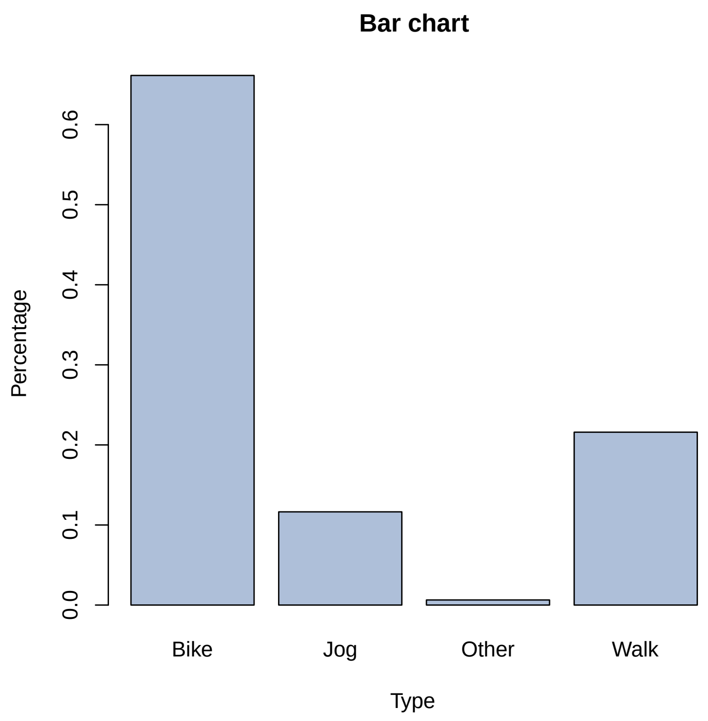


Figure 3.1 (using more advanced code) on page 61 of Utts and Heckard.

```
> pie(summary(Bike$Type), labels = paste(names(summary(Bike$Type)),
       round(100 * summary(Bike$Type)/length(Bike$Type), 1)))
> title("Pie chart")
```



```
> barplot(summary(Bike$Type)/length(Bike$Type))
> title("Bar chart", ylab = "Percentage", xlab = "Type")
```



The additional `t()` command transposes the table to match that given in Table 3.2. We need the first version when constructing Figure 3.2 which presents the *stack bar chart* and *cluster bar chart* for this data. The `addmargins()` command adds the row and column totals for the data given. This is because the default action for the `addmargins()` command is to provide these totals; other functions are possible.

Note that the `legend()` command needs to be issued explicitly so that we can see which bars represent each gender, and that unfortunately, R has chosen to use grey-scale in place of colours in the graphs. We can force the colours in graphs when this occurs — examples that might help follow later in this chapter. Also note the slight differences in the legends produced in these graphs.

Figure 3.3 is a little more difficult to generate as the `barplot()` command cannot directly handle three categorical variables. We trick R into dealing with two of the variables in combination using the `paste()` command which joins two text strings together.

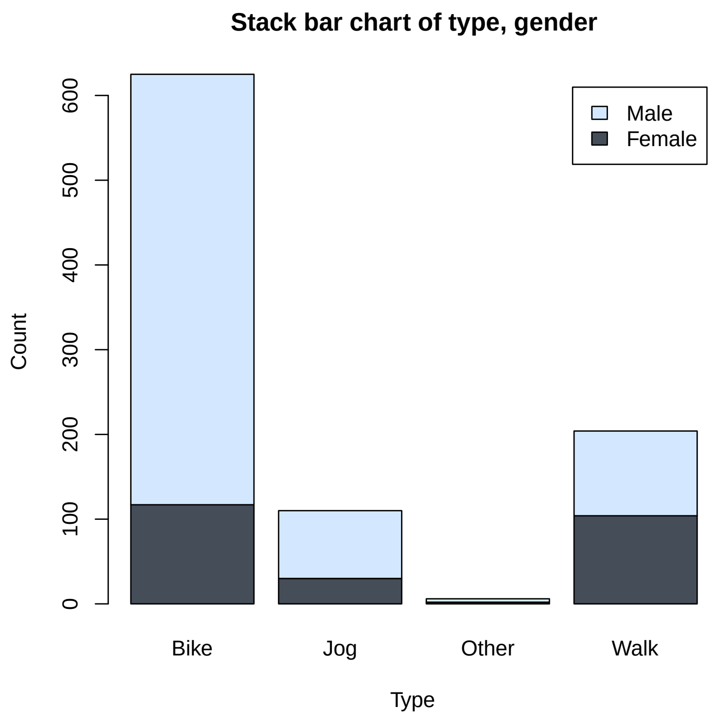
Creation of Table 3.3 on page 63 is not a simple reiteration of information already gathered. We can get the set of counts from work already done, but the correct amount to divide each count by is not quite so obvious.

```
> Counts = tapply(Bike$Time, list(Bike$Gender, Bike$Type),
       length)
> Counts

       Bike Jog Other Walk
Female  117  30     2  104
Male    508  80     4  100

> Divisor = matrix(tapply(Bike$Type, Bike$Type, length), nrow = 2,
       ncol = 4, byrow = T)
> Divisor

     [,1] [,2] [,3] [,4]
[1,]  625  110    6  204
[2,]  625  110    6  204

> round(t(100 * Counts/Divisor), 1)

      Female Male
Bike    18.7 81.3
Jog     27.3 72.7
Other   33.3 66.7
Walk    51.0 49.0
```

As before, R has not recognised that we probably want to put the 'Other' category last. It has processed the different types of transport in alphabetical order. We have also used the `round()` command here because we do not want an excessive number of decimal places being printed. We have also chosen not to undertake the presentation of the row totals for two reasons: First, they are somewhat obvious, and second, the R code needed to generate them and present them is well beyond any benefit gained.

Figure 3.2 on page 62 of Utts and Heckard.

```
> barplot(tapply(Bike$Time, list(Bike$Gender, Bike$Type), length),
      legend = TRUE)
> title("Stack bar chart of type, gender", ylab = "Count",
      xlab = "Type")
```



```
> barplot(tapply(Bike$Time, list(Bike$Gender, Bike$Type), length),
      legend = TRUE, beside = TRUE)
> title("Cluster bar chart of type, gender", ylab = "Count",
      xlab = "Type")
```

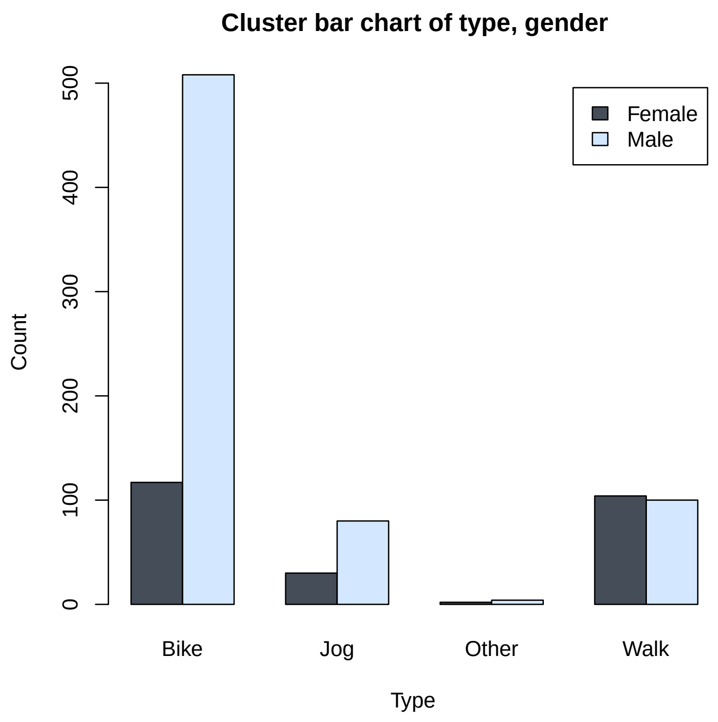
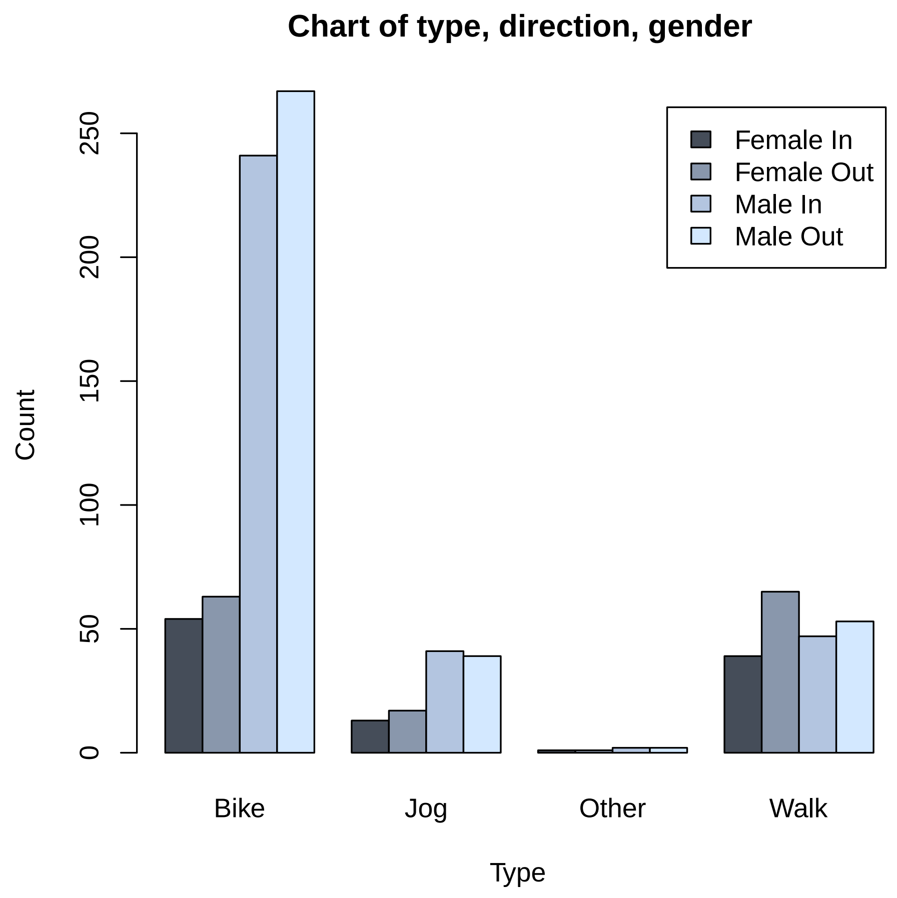


Figure 3.3 on page 62 of Utts and Heckard.

```
> barplot(tapply(Bike$Time, list(paste(Bike$Gender, Bike$Direction),
      Bike$Type), length), legend = TRUE, beside = TRUE)
> title("Chart of type, direction, gender", ylab = "Count",
      xlab = "Type")
```



In this first edition of this manual we have not reproduced the tables and figures that use the `PennState1` dataset which accompanied the United States edition of *Mind on Statistics*. The dataset is available as part of the *MindOnStats* package however, and could be obtained using:

```
> data(PennState1, package = "MindOnStats")
> str(PennState1)
'data.frame':       190 obs. of  9 variables:
 $ Sex     : Factor w/ 2 levels "Female","Male": 2 2 2 1 1 1 1 1 2 2 ...
 $ HrsSleep: num  5 7 6 7.5 7 3 5 9 4 9 ...
 $ SQpick  : Factor w/ 2 levels "Q","S": 2 2 2 2 2 2 2 2 2 2 ...
 $ Height  : num  67 75 73 64 63 65 64 68 73 69 ...
 $ RandNumb: int  3 9 7 8 7 2 7 8 10 8 ...
 $ Fastest : int  110 109 90 80 75 83 80 100 140 105 ...
 $ RtSpan  : num  21.5 22.5 23.5 20 19 ...
 $ LftSpan : num  21.5 22.5 24 21 19 ...
 $ Form    : Factor w/ 2 levels "QorS","SorQ": 2 2 2 2 2 2 2 2 2 2 ...
```

3.2 Graphs and plots for one continuous variable

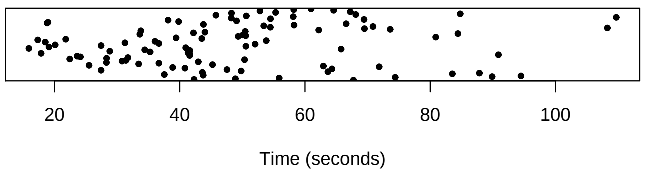
To create the graphs in this section we use the `HoldingBreath` dataset from the *MindOnStats* package.

```
> data(HoldingBreath, package = "MindOnStats")
> str(HoldingBreath)
'data.frame':       102 obs. of  14 variables:
 $ Gender   : Factor w/ 2 levels "f","m": 2 2 2 2 2 1 2 1 1 2 ...
 $ Age      : int  32 59 62 42 34 21 22 29 61 18 ...
 $ Size     : Factor w/ 4 levels "l","m","s","xl": 2 1 1 2 2 2 1 2 2 1 ...
 $ Smoker   : Factor w/ 2 levels "n","y": 2 1 2 2 1 2 1 2 1 2 ...
 $ Asthmatic: Factor w/ 2 levels "n","y": 1 1 1 1 1 2 1 2 1 1 ...
 $ Illness  : Factor w/ 2 levels "n","y": 1 1 1 1 1 2 1 1 2 1 ...
 $ Exercise : Factor w/ 2 levels "n","y": 1 1 1 2 2 2 2 1 1 2 ...
 $ Time     : num  67.7 42.3 48.9 55.9 74.4 ...
 $ C9.T     : logi  NA NA NA NA NA NA ...
 $ C10.T    : logi  NA NA NA NA NA NA ...
 $ C11.T    : logi  NA NA NA NA NA NA ...
 $ C12.T    : logi  NA NA NA NA NA NA ...
 $ C13.T    : logi  NA NA NA NA NA NA ...
 $ C14.T    : logi  NA NA NA NA NA NA ...
```

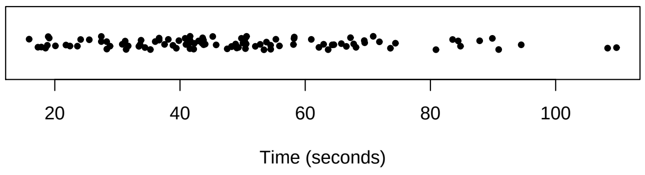
The way R generates a *dot plot* is different to other software. We present three different options in Figure 3.9 because we cannot reproduce the graph given in the text exactly without additional commands and a simple graph deserves to be created using the simplest code possible!

Figure 3.9 on page 70 of Utts and Heckard.

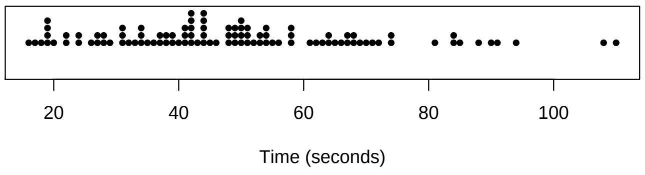
```
> windows(7, 2.5)
> dotchart(HoldingBreath$Time, pch = 20, xlab = "Time (seconds)",
       lcolor = 0)
```



```
> windows(7, 2.5)
> stripchart(HoldingBreath$Time, method = "jitter", pch = 20,
       xlab = "Time (seconds)")
```



```
> windows(7, 2.5)
> stripchart(round(HoldingBreath$Time, 0), method = "stack",
       pch = 20, xlab = "Time (seconds)")
```

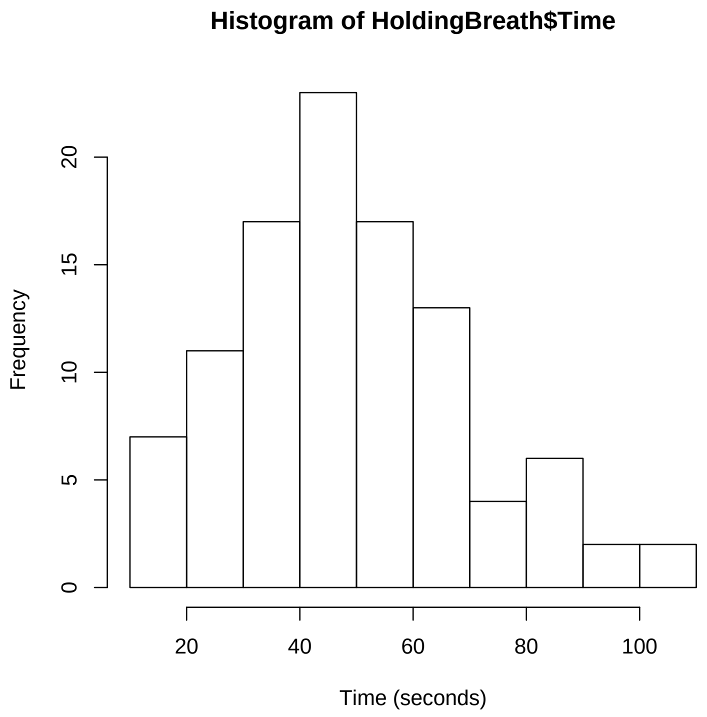


The `windows()` command opens a new graphics window. It is done automatically whenever we generate a graph and a window is not already waiting for our graph. When R does this, it chooses a default size, and more importantly a default aspect ratio for the incoming graph. The default really does not work well for the options for the *dot plot* shown here, or the *boxplot* graphs we see later in this section.

Figures 3.10 to 3.12 show the use of the `hist()`, `stem()`, and `boxplot()` commands.

Figure 3.10 on page 70 of Utts and Heckard.

```
> hist(HoldingBreath$Time, xlab = "Time (seconds)")
```



The histogram created by R automatically generates a main title. Change this using the `title()` command, or the `main` argument inside the `hist()` command itself. Note that the `stem()` command creates a textual display rather than a graph in a separate window. This is of course how all graphs were made at one time. The R version of the *stem-and-leaf plot* does not give the cumulative counts of data from top and bottom as does other software. R has also rounded the data to integers before creating the graph, and further, R has used two lines when other software uses one for each category. The differences are quite interesting to note.

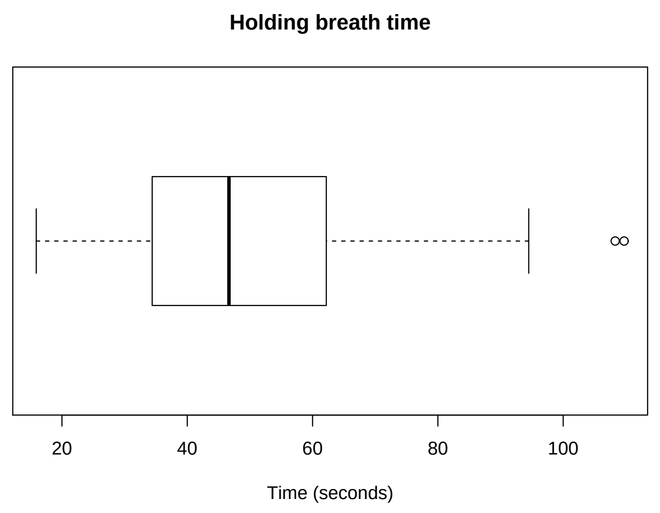
We have chosen to present the single *boxplot* as a horizontal graphic as it is slightly more attractive when only one boxplot is given.

Figure 3.11 on page 70 of Utts and Heckard.

```
> stem(HoldingBreath$Time)
  The decimal point is 1 digit(s) to the right of the |

   1 | 6789999
   2 | 02244
   2 | 677889
   3 | 11123444
   3 | 56778899
   4 | 011122222344444
   4 | 56888999
   5 | 0000111233444
   5 | 56888
   6 | 12344
   6 | 5677889
   7 | 01244
   7 |
   8 | 144
   8 | 58
   9 | 01
   9 | 5
  10 |
  10 | 8
  11 | 0
```

Figure 3.12 on page 71 of Utts and Heckard.

```
> windows(7, 5)
> boxplot(HoldingBreath$Time, horizontal = TRUE, xlab = "Time (seconds)",
       main = "Holding breath time")
```

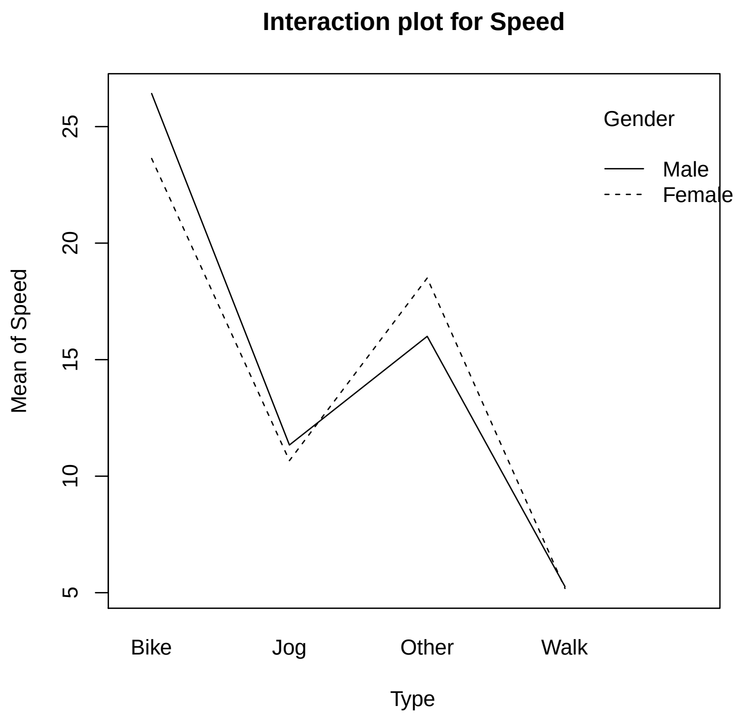


3.3 Continuous and categorical data

To generate Figure 3.16, we return to the `Bike` dataset. The `interaction.plot()` function uses the mean as its default action to take on the data for each combination of what are called the `x.factor` and `trace.factor`. We can change the function applied if we need to at a later date.

Figure 3.16 on page 74 of Utts and Heckard.

```
> interaction.plot(x.factor = Bike$Type, trace.factor = Bike$Gender,
       response = Bike$Speed, ylab = "Mean of Speed", xlab = "Type",
       trace.label = "Gender", main = "Interaction plot for Speed")
```



The legend is currently printing slightly outside the box for our graph. This can be fixed by adding the `inset=0.25`, but this generates a set of warning messages that might distress some novice users. It does, however, make the graph much tidier!

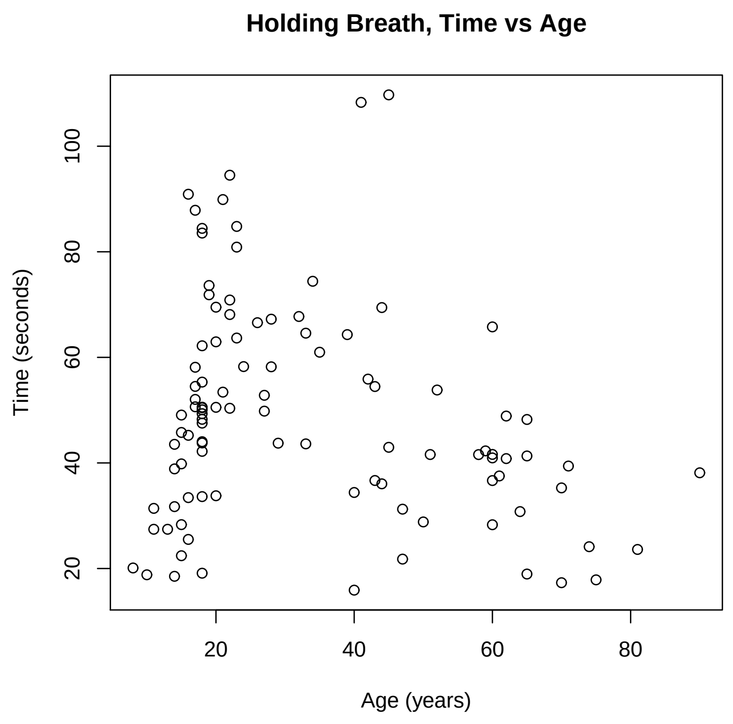
3.4 More than one continuous variable

By now you are probably tiring of typing the name of the dataset over and over again, so it's time to introduce the `attach()` command which attaches the dataset named so that we can directly access the variables within it. This is done in the next example.

In this section we make use of the `plot()` command which is a common command because it performs different actions for different objects. This is both confusing for some users and a joy for others working with R. It is used to create a *scatterplot* in Figure 3.17.

Figure 3.17 on page 75 of Utts and Heckard.

```
> attach(HoldingBreath)
> plot(x = Age, y = Time, xlab = "Age (years)", ylab = "Time (seconds)",
      main = "Holding Breath, Time vs Age")
> detach(HoldingBreath)
```



See that we've also used the `detach()` command in these figures. This undoes the `attach()` command. If you intend working with a single dataset for some time then you wouldn't really want to keep attaching and detaching it. The re-use of these commands is given here for completeness only, and because we swap datasets often enough that we might end up having two variables from different datasets having the same name; this does cause some conflicts between datasets. An alternative is to use the `data` argument within the `plot()` command, as done in the next example.

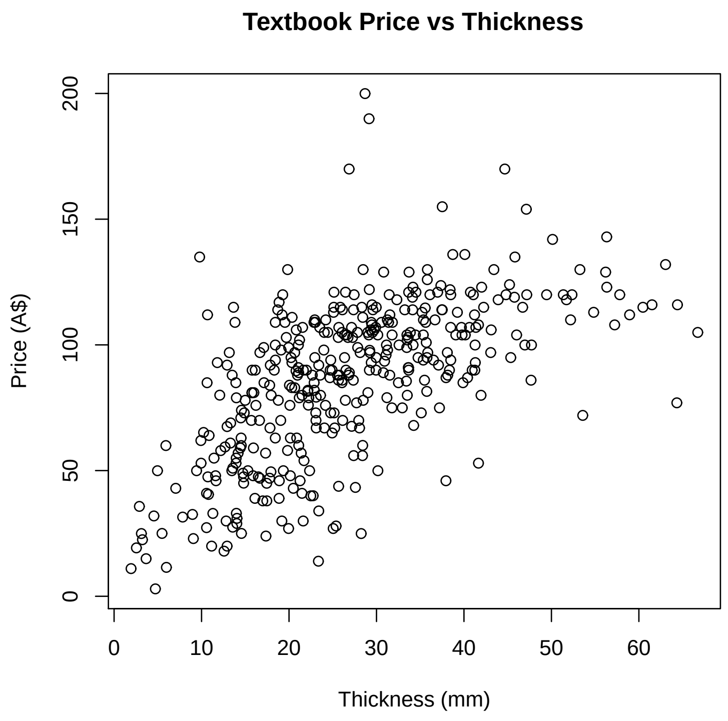
We now use another dataset, obtained via:

```
> data(Textbooks, package = "MindOnStats")
> str(Textbooks)
'data.frame':        394 obs. of  8 variables:
 $ Discipline: Factor w/ 14 levels "Accountancy",..: 8 8 8 8 8 8 8 8 8 8 ...
 $ Price     : num  48 11.5 100 78 19.9 ...
 $ Coverstyle: Factor w/ 2 levels "H","S": 2 2 2 1 2 2 2 2 2 2 ...
 $ Colour    : Factor w/ 2 levels "N","Y": 1 1 1 1 1 1 1 1 1 1 ...
 $ CD        : Factor w/ 2 levels "N","Y": 1 1 1 1 1 1 1 1 1 1 ...
 $ Thickness : num  11.6 5.98 32.58 28.48 11.15 ...
 $ Weight    : int  330 135 870 1085 220 645 195 1530 725 1130 ...
 $ Year      : int  2005 2003 2001 2006 2005 2007 2005 2005 2005 2007 ...
```

The `plot()` command is so flexible that it offers a number of different ways of getting the same outcome. In Figure 3.18, we use the formula approach to define the two variables to be plotted. This use of `Price~Thickness` uses the tilde and is common to many R commands. We also use the `data` argument here for illustration.

Figure 3.18 on page 76 of Utts and Heckard.

```
> plot(Price ~ Thickness, data = Textbooks, xlab = "Thickness (mm)",
      ylab = "Price (A$)", main = "Textbook Price vs Thickness")
```



This figure is created using a set of commands. First we create the axes for the graph using the `plot()` command without actually plotting any data, using the `type` argument. Then we use the `points()` command to add points to the graph for the male respondents by selecting the two variables for only the male subjects using the subscript notation; the points for the female respondents are added in similar fashion. Note the use of both the type of symbol plotted and the colour used in this graph. This was done for the benefit of both black-and-white print users and those readers viewing the electronic copy which benefits from the use of colour. Finally, we add a `legend()` command for obvious reasons which uses the same information as was given to the `points()` commands it follows. Finally, see we reverted to the use of `attach()` and `detach()` commands which was a more efficient way of putting this code together.

Figure 3.20 on page 77 of Utts and Heckard.

```
> attach(HoldingBreath)
> plot(x = Age, y = Time, xlab = "Age (years)", ylab = "Time (seconds)",
      main = "Holding Breath, Time vs Age", type = "n")
> points(x = Age[Gender == "m"], y = Time[Gender == "m"], pch = 21,
      col = 4)
> points(x = Age[Gender == "f"], y = Time[Gender == "f"], pch = 19,
      col = 2)
> legend("topright", title = "Gender", legend = c("Male", "Female"),
      pch = c(21, 19), col = c(4, 2))
> detach(HoldingBreath)
```

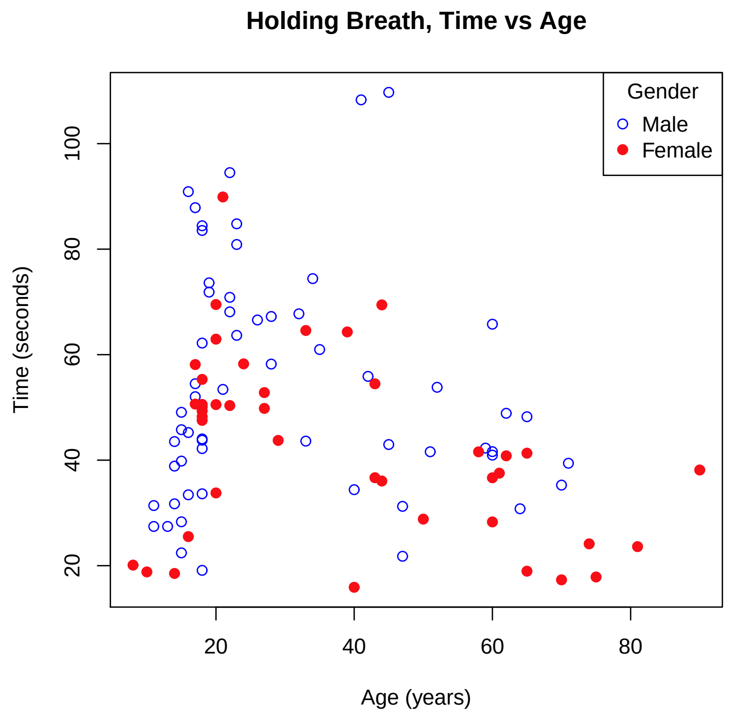
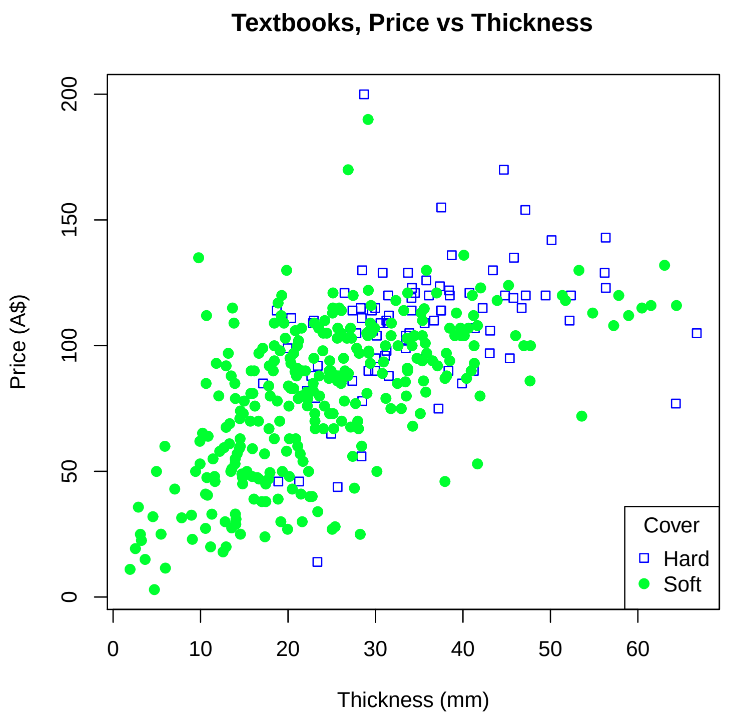


Figure 3.21 has a similar structure to Figure 3.20 but we have used more complete code for such items as the colours used. It's personal preference as to the use of words vs numbers, but you need to note the use of quote marks around the text information provided to some commands.

Figure 3.21 on page 78 of Utts and Heckard.

```
> attach(Textbooks)
> plot(Price ~ Thickness, xlab = "Thickness (mm)", ylab = "Price (A$)",
      main = "Textbooks, Price vs Thickness", type = "n")
> points(Price[Coverstyle == "H"] ~ Thickness[Coverstyle ==
      "H"], pch = 22, col = "blue")
> points(Price[Coverstyle == "S"] ~ Thickness[Coverstyle ==
      "S"], pch = 19, col = "green")
> legend("bottomright", title = "Cover", legend = c("Hard",
      "Soft"), pch = c(22, 19), col = c("blue", "green"))
> detach(Textbooks)
```

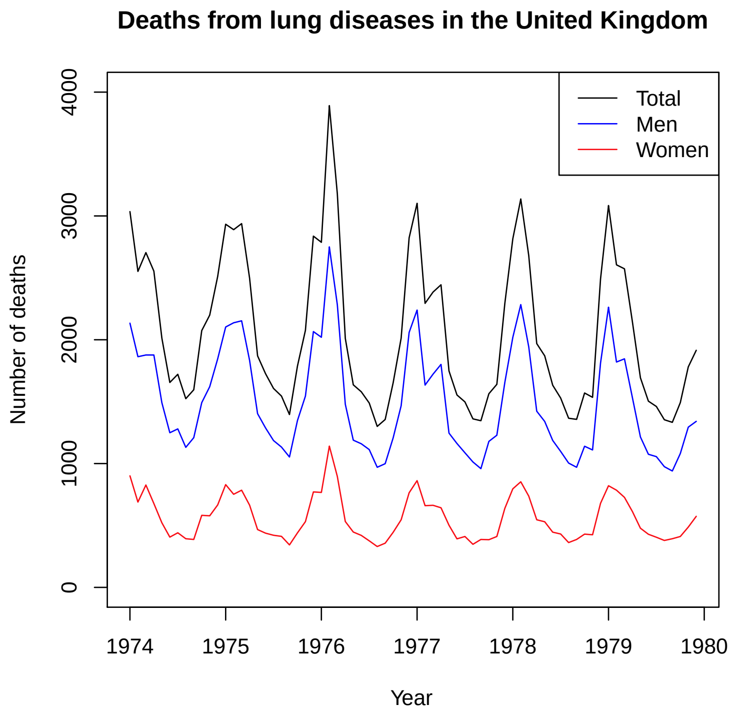


None of the datasets provided with the textbook lend themselves to being plotted as a *time series plot*. We have found a dataset that is provided with the default installation of R to illustrate this type of plot. Note that we have used the `plot()` command again, and have specified only a single variable to be plotted in the first of the three commands needed.

Notice that we have forced R to show the range of values spanned by the y-axis using the `ylim` argument to the `plot()` command. We have then added the data for two more series using the `lines()` command which does the same job as the `points()` command used earlier

Extra Figure 1 is not a replication of work given in Utts and Heckard.

```
> plot(ldeaths, ylim = c(0, 4000), xlab = "Year", ylab = "Number of deaths",
       main = "Deaths from lung diseases in the United Kingdom")
> lines(fdeaths, col = "red")
> lines(mdeaths, col = "blue")
> legend("topright", legend = c("Total", "Men", "Women"), col = c("black",
       "blue", "red"), lty = c(1, 1, 1))
```



except for the change from points to a series of line segments.

3.5 Outliers

There are no graphs, tables, or calculations requiring R in this section.

3.6 Good graphs and bad graphs

There are no graphs, tables, or calculations requiring R in this section.

Chapter 4

Data features and summary statistics

4.1 Commenting on features of data

We use the `Bike` data again to extend some graph techniques shown in the previous chapter.

```
> data(Bike, package = "MindOnStats")
```

The first task is to extract only those users of the bikeway that are actually using a bike. We create a second dataset called `Bike2`, based on the first:

```
> Bike2 = Bike[Bike$Type == "Bike", ]
> str(Bike2)
'data.frame':        625 obs. of  5 variables:
 $ Time     : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Type     : Factor w/ 4 levels "Bike","Jog","Other",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ Speed    : int  30 36 26 21 28 37 27 34 21 29 ...
 $ Direction: Factor w/ 2 levels "In","Out": 1 1 1 1 1 1 1 1 1 1 ...
 $ Gender   : Factor w/ 2 levels "Female","Male": 2 2 2 2 2 2 2 2 2 2 ...
```

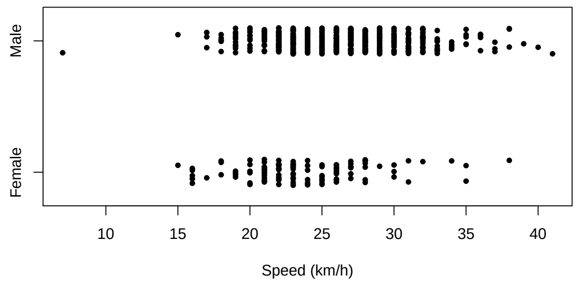
We have presented the two options for using the `stripchart()` command in Figure 4.1. Use of the `windows()` command can have an impact on the presentation of these dot plots. We leave it to the user to alter the height of the graph window to suit their personal preferences.

We need the `Mobiles` dataset for creating Figure 4.3. It is the first time we have used this dataset so we import it from the *MindOnStats* package and take a quick look at its structure using:

```
> data(Mobiles, package = "MindOnStats")
> str(Mobiles)
'data.frame':        179 obs. of  10 variables:
 $ Gender    : Factor w/ 2 levels "Female","Male": 1 2 1 2 1 2 2 2 2 2 ...
 $ Age       : int  17 21 20 23 38 19 23 24 19 19 ...
 $ Faculty   : Factor w/ 10 levels "Arts","Built Env",..: 1 1 1 1 1 2 2 2 2 2 ...
 $ Brand     : Factor w/ 15 levels "","Alcatel","Ericsson",..: 4 6 12 1 8 3 3 3 6 6 ...
```

Figure 4.1 on page 94 of Utts and Heckard.

```
> windows(7, 4)
> stripchart(Speed ~ Gender, data = Bike2, method = "jitter",
       pch = 20, xlab = "Speed (km/h)")
```



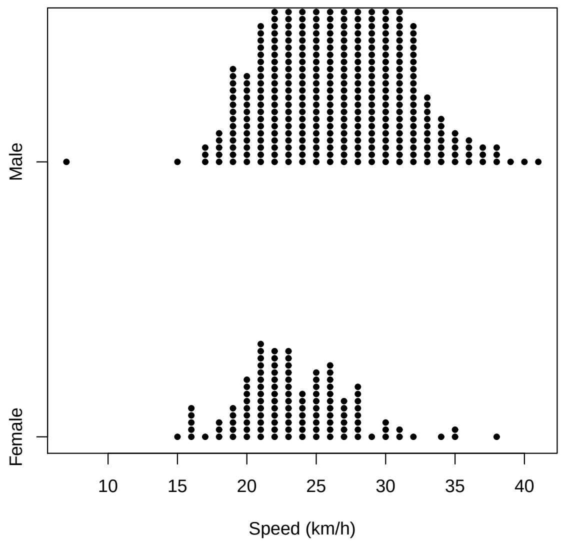
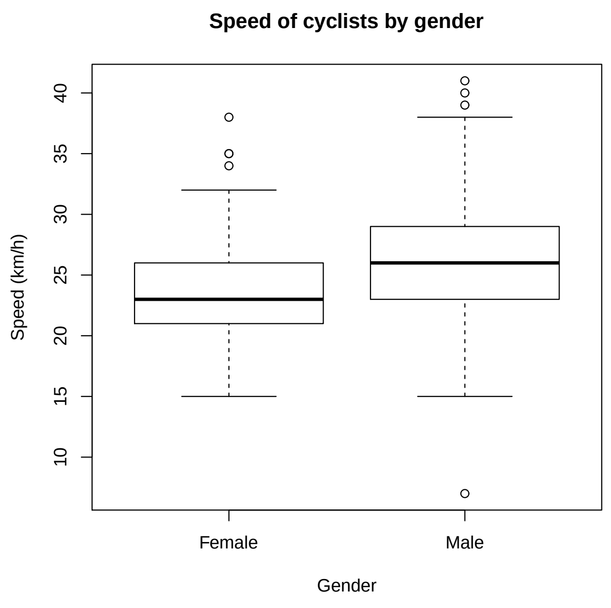
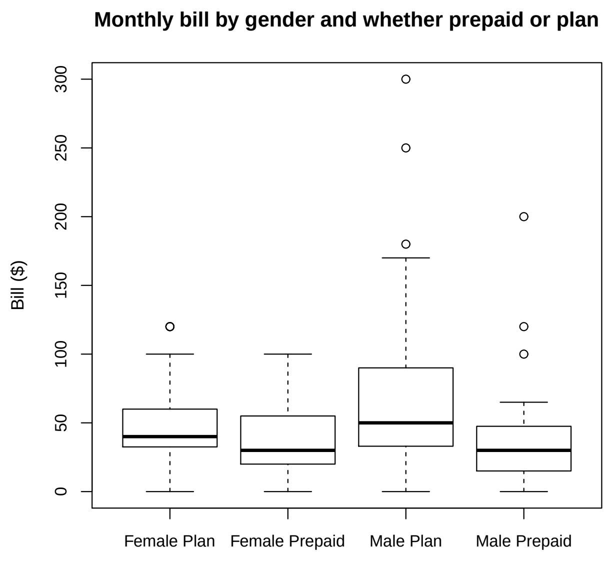


Figure 4.2 on page 94 of Utts and Heckard.



```
$ Colour    : Factor w/ 13 levels "","Black","Blue",..: 12 12 12 1 3 3 2 12 3 7 ...
$ Provider  : Factor w/ 7 levels "","3","Optus",..: 5 4 3 1 3 7 5 3 7 5 ...
$ PlanType  : Factor w/ 3 levels "","Plan","Prepaid": 2 3 3 1 2 3 2 2 3 2 ...
$ Bill      : int  70 200 30 NA 120 10 100 30 20 43 ...
$ PrimaryUse: Factor w/ 4 levels "","Both","Calls",..: 4 3 2 1 3 2 2 3 4 3 ...
$ No.Phones : int  1 3 1 2 3 1 1 9 1 4 ...
```

Figure 4.3 on page 95 of Utts and Heckard.



We have needed to use the graphical parameter `cex.axis` when we produced this graph. This expands the size of the text on the axis markers of the graph, and may not be necessary on your configuration of R. Experiment and see if it is necessary.

4.2 Measures of location

The `Fish` dataset is used in this section. Obtain it using:

```
> data(Fish, package = "MindOnStats")
> str(Fish)
'data.frame':        58 obs. of  4 variables:
 $ FishType: Factor w/ 9 levels "bream        ",..: 1 1 1 1 1 1 3 3 3 3 ...
 $ Length  : int  170 250 360 410 245 230 200 270 240 250 ...
 $ Weight  : int  250 395 720 890 390 380 310 465 370 380 ...
 $ Angler  : int  1 1 1 2 2 1 2 2 1 2 ...
```

Note the use of a capital letter at the start of the variable names. It is important that a variable name is not confused with R commands. Most R commands are lower case and we have converted all variable names in the supplied datasets to start with capitals to avoid any clashes. The `Length` variable is therefore not going to be confused with the `length()` command.

We use the `sort()` command to put the lengths of the fish in ascending order.

```
> SortedLengths = sort(Fish$Length)
> SortedLengths
 [1]   85   92  110  170  170  180  185  185  185  200  230  230  230  235
[15]  240  240  240  240  240  245  245  250  250  250  250  250  250  250
[29]  250  260  270  270  270  270  270  270  280  280  280  280  285  290
[43]  290  305  310  315  320  330  340  345  350  360  370  385  390  400
[57]  410 1200
```

Then we use the `mean()` and `median()` commands to get two measures of location:

```
> mean(Fish$Length)
[1] 278.8
> median(Fish$Length)
[1] 255
```

Note the `summary()` command also extracts these measures of location, but it gives us other measures that might not be needed at this time.

4.3 Measures of spread

The reef shark is dropped because it is a much longer fish than the others in the dataset.

```
> length(SortedLengths)
[1] 58
> FishLengthsNoShark = SortedLengths[-58]
```

Note the subscripting has used the subtraction operator to subtract the 58th observation from the set of sorted lengths.

Now, using the `FishLengthsNoShark` variable we have created, we can find the various measures of spread using the `range()`, `sd()` (standard deviation), `IQR()` (interquartile range), and `var()` (variance) commands.

```
> range(FishLengthsNoShark)
[1]  85 410
> sd(FishLengthsNoShark)
[1] 69.4
> IQR(FishLengthsNoShark)
[1] 50
> var(FishLengthsNoShark)
[1] 4816
```

There are several ways to generate the quartiles for this data. The `summary()` command applied to a variable will generate them, the `fivenum()` command will, and we can also get them using the `quantile()` command. The differences are briefly explained in the help for `quantile()` command, where we learn that there are at least nine different versions. We present the three approaches, but others exist:

```
> fivenum(FishLengthsNoShark)
[1]  85 240 250 290 410
> summary(FishLengthsNoShark)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
     85     240     250     263     290     410
> quantile(FishLengthsNoShark, probs = c(0.25, 0.75))
25% 75%
240 290
```

4.4 Shape

To take the natural logarithm of a set of numbers, we apply the `log()` command.

```
> lnBills = log(Mobiles$Bill)
```

To replace the EXCEL tip on page 102, we show the use of the `min()`, `max()`, and `quantile()` commands applied to the `FishLengthsNoShark` data. To calculate the 90^{th} percentile:

```
> quantile(FishLengthsNoShark, probs = 0.9)
90%
354
```

and the minimum and maximum values:

```
> min(FishLengthsNoShark)
[1] 85
> max(FishLengthsNoShark)
[1] 410
```

The other functions explained in this tip have been introduced already, namely: `length()` for counts and others given in this chapter.

4.5 Parameters, models and estimates

There are no examples in this section requiring use of `R`.

Chapter 5

Investigating categorical variables and their relationships

5.1 Summarising and presenting categorical data

We use the `PopSong` dataset for a number of examples in this chapter. Obtain it using:

```
> data(PopSong, package = "MindOnStats")
> str(PopSong)
'data.frame':        763 obs. of  8 variables:
 $ Year     : int  1998 1998 1998 1998 1998 1998 1998 1998 1998 1998 ...
 $ Position : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Chart    : Factor w/ 2 levels "ARIA","JJJ": 1 1 1 1 1 1 1 1 1 1 ...
 $ Length   : int  272 248 289 295 276 180 199 154 213 260 ...
 $ Genre    : Factor w/ 8 levels "AltRock","Dance",..: 2 3 8 8 6 1 6 8 6 6 ...
 $ Country  : Factor w/ 4 levels "Australia","Other",..: 2 4 4 4 3 1 4 4 2 2 ...
 $ Publisher: Factor w/ 9 levels "BMG","EMI","Festival",..: 6 8 6 6 7 2 5 8 7 7 ...
 $ Author   : Factor w/ 3 levels "Band","Cover",..: 3 1 1 1 3 1 3 1 3 3 ...
```

To re-create the information presented in Table 5.1 and Figure 5.1, we use slightly more concise code than was used in previous chapters. First we need to extract only the data for 2001:

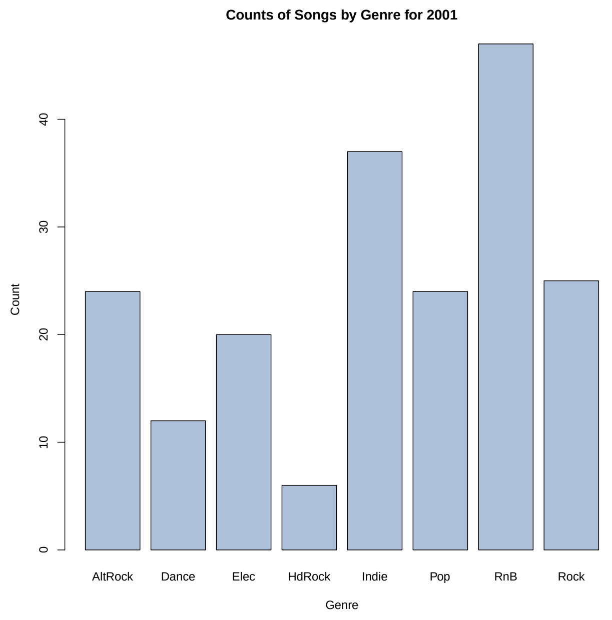
```
> PopSong01 = PopSong[PopSong$Year == 2001, ]
> str(PopSong01)
'data.frame':        195 obs. of  8 variables:
 $ Year     : int  2001 2001 2001 2001 2001 2001 2001 2001 2001 2001 ...
 $ Position : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Chart    : Factor w/ 2 levels "ARIA","JJJ": 1 1 1 1 1 1 1 1 1 1 ...
 $ Length   : int  217 231 230 235 215 404 243 209 190 199 ...
 $ Genre    : Factor w/ 8 levels "AltRock","Dance",..: 6 7 2 7 8 7 6 8 2 6 ...
 $ Country  : Factor w/ 4 levels "Australia","Other",..: 4 4 1 4 4 4 4 4 3 3 ...
 $ Publisher: Factor w/ 9 levels "BMG","EMI","Festival",..: 6 7 3 7 7 7 7 7 7 2 ...
 $ Author   : Factor w/ 3 levels "Band","Cover",..: 1 1 3 3 1 1 1 2 3 3 ...
```

Then using the `tapply()` command to find the contents of the various elements of the table, and the `plot()` command to generate the bar chart in Figure 5.1:

```
> attach(PopSong01)
> tapply(Genre, Genre, length)
AltRock   Dance    Elec  HdRock   Indie     Pop     RnB    Rock
     24      12      20       6      37      24      47      25
> length(Genre)
[1] 195
> tapply(Genre, Genre, length)/length(Genre)
AltRock   Dance    Elec  HdRock   Indie     Pop     RnB    Rock
0.12308 0.06154 0.10256 0.03077 0.18974 0.12308 0.24103 0.12821
```

Figure 5.1 on page 118 of Utts and Heckard.

```
> plot(Genre, xlab = "Genre", ylab = "Count", main = "Counts of Songs by Genre for 2001")
```



```
> detach(PopSong01)
```

Note the convenience of accessing the variable names directly having used the `attach()` and `detach()` commands here.

5.2 More than two categorical variables

The `aggregate()` command is a good way to obtain a table of results where there are combinations of more than two categorical variables that classify responses. The following example is not given in Utts and Heckard but is given here to illustrate the `aggregate()` command.

```
> aggregate(Position ~ Chart + Country + Author, data = PopSong,
       FUN = length)
   Chart   Country Author Position
1   ARIA Australia   Band       39
2    JJJ Australia   Band      163
3   ARIA     Other   Band       31
4    JJJ     Other   Band       20
5   ARIA        UK   Band       23
6    JJJ        UK   Band       52
7   ARIA       USA   Band      108
8    JJJ       USA   Band      122
9   ARIA Australia  Cover        1
10   JJJ Australia  Cover        4
11  ARIA     Other  Cover        1
12   JJJ     Other  Cover        1
13  ARIA        UK  Cover        9
14  ARIA       USA  Cover       18
15   JJJ       USA  Cover        5
16  ARIA Australia Songwr       19
17  ARIA     Other Songwr       40
18  ARIA        UK Songwr       44
19  ARIA       USA Songwr       63
```

The `FUN` argument tells the `aggregate()` command to apply the `length()` command to each combination of the factors described on the right-hand-side of the formula given. The range of functions that can be supplied is pretty much the same as for the `tapply()` command seen already.

5.3 One categorical variable: testing a set of proportions

The χ^2-distribution is used in this section. To find a probability for a particular value from this distribution, we use the `pchisq()` command. To replicate the result shown in the Minitab tip on page 127 use:

```
> pchisq(5.935, df = 2, lower.tail = FALSE)
[1] 0.05143
```

The Minitab tip on page 129 showing how to perform a goodness of fit test, can be replicated using the `chisq.test()` command:

```
> chisq.test(x = c(42, 49, 69), p = c(0.22, 0.4, 0.38))
        Chi-squared test for given probabilities

data:  c(42, 49, 69)
X-squared = 5.935, df = 2, p-value = 0.05143
```

5.4 p-values and testing statistical hypotheses

There are no examples in this section requiring R.

5.5 Testing independence of two categorical variables

We need to see the contingency table for the Genre and Chart variables in the `PopSong` dataset. The `tapply()` command does this for us:

```
> attach(PopSong)
> tapply(Position, list(Genre, Chart), length)
        ARIA JJJ
AltRock    5  89
Dance     78   2
Elec      15  44
HdRock     9  26
Indie      7 151
Pop      141   3
RnB       87  14
Rock      54  38
> detach(PopSong)
```

Then to combine the two categories and re-create the table we use:

```
> attach(PopSong)
> NewGenre = Genre
> levels(NewGenre)
[1] "AltRock" "Dance"   "Elec"    "HdRock"  "Indie"   "Pop"     "RnB"
[8] "Rock"
> levels(NewGenre)[1] = "AnHRock"
> levels(NewGenre)[4] = "AnHRock"
> levels(NewGenre)
[1] "AnHRock" "Dance"   "Elec"    "Indie"   "Pop"     "RnB"     "Rock"
> tapply(Position, list(NewGenre, Chart), length)
        ARIA JJJ
AnHRock   14 115
Dance     78   2
Elec      15  44
Indie      7 151
Pop      141   3
RnB       87  14
Rock      54  38
> detach(PopSong)
```

We will use the `chisq.test()` command to test the notion that a pair of categorical variables are independent of one another. We test the independence of the `NewGenre` variable just created, and the `Chart` variable from the original dataset using:

```
> chisq.test(NewGenre, PopSong$Chart)
        Pearson's Chi-squared test

data:  NewGenre and PopSong$Chart
X-squared = 484.2, df = 6, p-value < 2.2e-16
```

Case Study 5.1 uses Table 5.20 which appears on page 135. We enter it using the `c()` and `matrix()` commands. For completeness, we also add in the row and column names using the `dimnames()` command.

```
> BreastCancer = matrix(c(166, 8340, 124, 7978), nrow = 2,
       byrow = TRUE)
> dimnames(BreastCancer)[[1]] = c("Hormones", "Placebo")
> dimnames(BreastCancer)[[2]] = c("Yes", "No")
> BreastCancer
         Yes   No
Hormones 166 8340
Placebo  124 7978
```

We then perform a test for the independence of the two factors (row and column) using the `chisq.test()`:

```
> chisq.test(BreastCancer)
        Pearson's Chi-squared test with Yates' continuity correction

data:  BreastCancer
X-squared = 4.046, df = 1, p-value = 0.04426
```

5.6 Risk, relative risk and misleading statistics about risk

No additional R commands need to be introduced in this section.

Chapter 6

Probability essentials for data analysis

There are few tasks in this chapter that are beyond simple mathematical operations. R could of course be used as a basic calculator in these instances.

6.1 What is probability?

We do not present any examples from this section using R.

6.2 Where do values of probabilities come from?

We do not present any examples from this section using R.

6.3 What is a random variable?

We do not present any examples from this section using R.

6.4 Expected values and standard deviations of random variables

We do not present any examples from this section using R.

6.5 Three special distributions

The `cumsum()` command takes a set of numbers (called a vector by R) and returns the cumulative sum of all objects from the first up to each point. For example:

```
> x = 1:5
> x
```

```
[1] 1 2 3 4 5
> cumsum(x)
[1]  1  3  6 10 15
```

This functionality is embedded in some commands that we introduce in this section.

Example 6.17 on page 174 seeks a probability using the binomial distribution. We find these using the `dbinom()` command, where the `d` stands for density.

```
> dbinom(7, size = 10, prob = 0.487)
[1] 0.1053
```

The arguments for the `dbinom()` command, and the related commands for other binomial distribution actions, use `size` and `prob` as the way of providing the two parameters of the binomial distribution.

To accumulate a set of binomial probabilities it is probably easiest to use the `dbinom()` command and add up the results using the `sum()` command. For example, in Example 6.18 where we want the sum of the probabilities of getting 10 or more out of 15 attempts, we could:

```
> x = 10:15
> dbinom(x, size = 15, prob = 0.5)
[1] 9.164e-02 4.166e-02 1.389e-02 3.204e-03 4.578e-04 3.052e-05
> sum(dbinom(x, size = 15, prob = 0.5))
[1] 0.1509
```

or use the `pbinom()` command which does the accumulation for us:

```
> pbinom(9, size = 15, prob = 0.5, lower.tail = FALSE)
[1] 0.1509
```

but when we do use the `pbinom()` command we need to remember that the function gives us the cumulative distribution value up to the point specified, or its complement if the `lower.tail` argument is set to `FALSE`.

Turning to the normal distribution, we use the `pnorm()` command to obtain probabilities. This evaluates the cumulative distribution function at a specified point. We rework the elements of Example 6.20 on page 180 using:

```
> pnorm(25, mean = 26.42, sd = 4.3)
[1] 0.3706
```

for a situation where we want the probability of being less than 25km/h. Then we add the `lower.tail=FALSE` argument to get the probability of being greater than 30km/h:

```
> pnorm(30, mean = 26.42, sd = 4.3, lower.tail = FALSE)
[1] 0.2025
```

and finally, we need to get the two probabilities required to find the probability of being within the range 25 to 30 km/h:

```
> pnorm(c(25, 30), mean = 26.42, sd = 4.3)
[1] 0.3706 0.7975
> diff(pnorm(c(25, 30), mean = 26.42, sd = 4.3))
[1] 0.4268
```

The `diff()` command gives us a convenient shortcut to subtract the first result from the second.

6.6 Normal probability plots

We use the `qqnorm()` command to generate the points plotted on a normal probability plot, then we add a straight line to the plot to help us determine the normality of the data using the `qqline()` command.

Figure 6.12 presents the normal probability plot for the speeds of the male cyclists in the `Bike` dataset.

```
> data(Bike, package = "MindOnStats")
> MaleCyclists = Bike$Speed[Bike$Gender == "Male" & Bike$Type ==
       "Bike"]
```

See how we have incorporated two conditions for the observations of `Speed` to be included in the data to be analysed here. R does not present the intervals given by Minitab.

The `TimePerception` dataset is used for the first time in this section. See Figure 6.13 for a normal probability plot on one of its variables, and Figure 6.14 for another. In both cases, histograms are also presented. The data are obtained using:

```
> data(TimePerception, package = "MindOnStats")
> str(TimePerception)
'data.frame':        120 obs. of  4 variables:
 $ FiveSec : num  3.62 5.05 4.24 4.35 4.01 ...
 $ TenSec  : num  6.73 9.51 5.68 7.34 8.2 ...
 $ Gender  : Factor w/ 2 levels "female","male  ": 1 1 1 1 1 1 1 1 1 1 ...
 $ AgeGroup: Factor w/ 6 levels "10-20","21-30",..: 1 1 1 1 1 1 1 1 1 1 ...
```

This dataset is used again in Chapter 9 on regression.

Figure 6.12 on page 184 of Utts and Heckard.

```
> qqnorm(MaleCyclists)
> qqline(MaleCyclists)
```

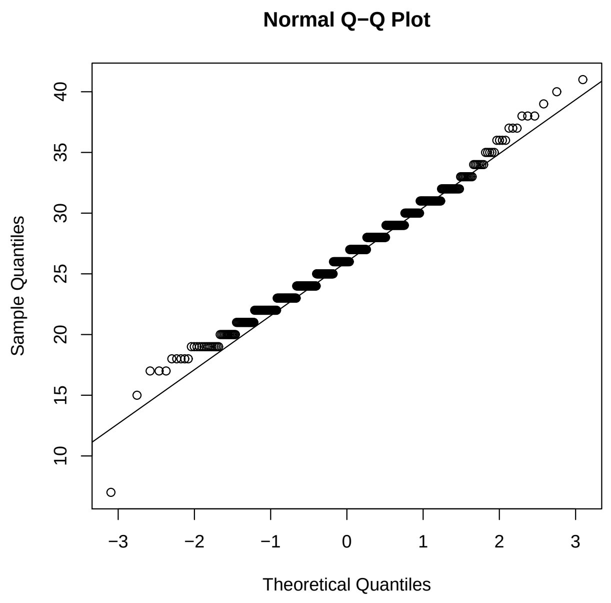


Figure 6.13 on page 184 of Utts and Heckard.

```
> attach(TimePerception)
> hist(FiveSec)
```

```
> qqnorm(FiveSec)
> qqline(FiveSec)
> detach(TimePerception)
```

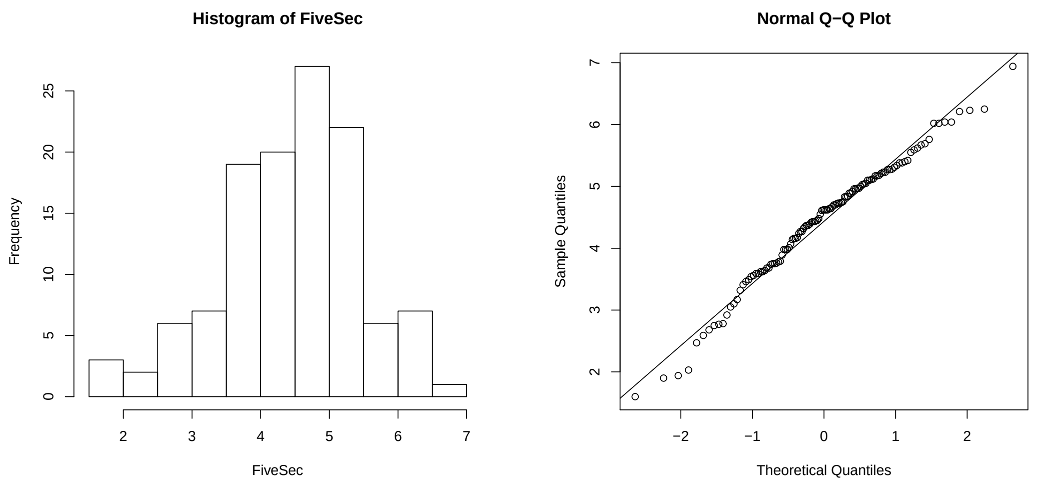
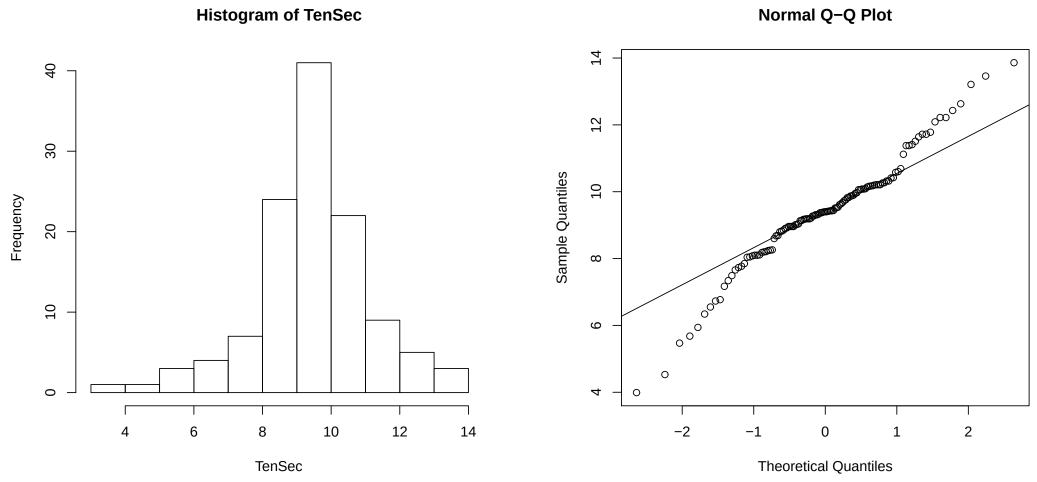


Figure 6.14 on page 185 of Utts and Heckard.

```
> attach(TimePerception)
> hist(TenSec)


> qqnorm(TenSec)
> qqline(TenSec)
> detach(TimePerception)
```



Chapter 7

Estimating proportions with confidence

7.1 Percentages and proportions abound

We do not present any examples from this section using R.

7.2 Confidence intervals for proportions

The `prop.test()` command provides a confidence interval for a single proportion. The problem is that it is not the same calculation as the options given by Utts and Heckard's TAM or AAM approaches. The JEM approach is obtainable using the `binom.test()` command, for example:

```
> binom.test(9, 235)
        Exact binomial test

data:  9 and 235
number of successes = 9, number of trials = 235, p-value < 2.2e-16
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.01766 0.07145
sample estimates:
probability of success
                0.0383
```

but we are given more infomation that we want at this time — very unusual for R! To calculate the TAM and AAM based confidence intervals for the same scenario, we suggest using basic calculations.

```
> p = 9/235
> s = sqrt(p * (1 - p)/235)
> z = c(-1.96, 1.96)
> p + z * s
```

```
[1] 0.01376 0.06284
```

where we have chosen the z-values for the 95% confidence interval ourselves. For the AAM approach:

```
> p = 11/239
> s = sqrt(p * (1 - p)/239)
> p + z * s
[1] 0.01946 0.07259
```

Just to illustrate the fact that the `prop.test()` command output is different:

```
> prop.test(9, 235)
        1-sample proportions test with continuity correction

data:  9 out of 235, null probability 0.5
X-squared = 198.5, df = 1, p-value < 2.2e-16
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.01880 0.07387
sample estimates:
     p
0.0383
```

7.3 Background for confidence intervals for proportions

Most of the tasks required in this section have been covered elsewhere so we only show the additional commands that might be useful.

To generate a set of random numbers that follow a binomial distribution we use the `rbinom()` command.

```
> x = rbinom(n = 1000, size = 100, prob = 0.6)
```

This generates a vector of length 1000 from a binomial distribution with $n = 100$, and $p = 0.6$.

7.4 Confidence intervals for the difference in two proportions

The `prop.test()` command can be used to provide a confidence interval for the difference in two proportions. The choice not to use Yates' correction gives results as close to those presented in Utts and Heckard as possible.

```
> prop.test(x = c(210, 213), n = c(432, 513), correct = FALSE)
```

```
        2-sample test for equality of proportions without continuity
        correction

data:  c(210, 213) out of c(432, 513)
X-squared = 4.769, df = 1, p-value = 0.02898
alternative hypothesis: two.sided
95 percent confidence interval:
 0.007349 0.134464
sample estimates:
prop 1 prop 2
0.4861 0.4152
```

The one reassuring fact from using this command is that it gives the same results as those given by Minitab on page 216.

7.5 Confidence intervals and decisions

No extra work in this section requires further use of R.

7.6 Sample size to estimate a proportion

We do not present any examples from this section using R as there are only a few basic calculations. The `power.prop.test()` command might be worth investigating after consideration of *power*. See Section 11.13 for more detail.

Chapter 8

Analysis of variance: investigating effects of categorical variables on a continuous variable

8.1 Some examples of data investigations that include continuous responses and possible categorical explanatory variables

No examples in this section require use of R.

8.2 One-way ANOVA

We obtain the GoGoGo dataset by issuing the data() command as before:

```
> data(GoGoGo, package = "MindOnStats")
> str(GoGoGo)
```

```
'data.frame':        100 obs. of  7 variables:
 $ Lights  : Factor w/ 2 levels "amber","green": 2 2 2 2 2 2 2 2 2 2 ...
 $ AgeGroup: Factor w/ 3 levels "<30",">40","30-40": 2 3 3 3 3 3 3 1 1 3 ...
 $ Gender  : Factor w/ 2 levels "F","M": 2 2 1 2 2 1 2 2 2 2 ...
 $ Colour  : Factor w/ 9 levels "BLACK","BLUE",..: 7 2 5 6 9 7 4 8 6 9 ...
 $ Type    : Factor w/ 8 levels "4WD","BUS","LARGE",..: 3 3 3 4 6 3 3 6 8 8 ...
 $ Make    : Factor w/ 18 levels "BMW","DAEWOO",..: 6 4 17 16 6 12 13 6 4 4 ...
 $ Time    : num  5.17 3.52 4.1 3.9 4.19 3.75 4.8 4.64 4.65 3.65 ...
```

The aov() command is used for many *analysis of variance* models. It provides the simplest results via the associated summary() command.

To replicate Example 8.6, we need to extract the data for only those observations that involve a green light. The `aov()` command has two useful arguments that we will take advantage of throughout this chapter.

```
> GreenLights.aov = aov(Time ~ AgeGroup, data = GoGoGo, subset = Lights ==
      "green")
> summary(GreenLights.aov)
            Df Sum Sq Mean Sq F value Pr(>F)
AgeGroup     2   5.16   2.580    4.94  0.011 *
Residuals   47  24.52   0.522
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The `data` argument in this `aov()` command gives us direct access to the names of variables in the formula statement and the `subset` argument. The `subset` command gives us the chance to extract only those rows of the dataset that conform to a particular constraint. Note on this occasion that we want only the drivers heading for a green light and that a double equals sign is required. Use of `==` distinguishes this relationship from the use of the single equals sign which assigns a value to an argument within a command.

For the Example 8.7 we use the `Textbooks` dataset. Obtain it using:

```
> data(Textbooks, package = "MindOnStats")
```

Notice how the following command combines the two commands introduced in the last example into a single line of code. This isn't necessary and some R users would prefer the previous approach. Also note the use of the `<-` which is for an assignment. The right-hand-side of this relationship is assigned to the object named on the left.

```
> summary(Price.aov1 <- aov(Price ~ Discipline, data = Textbooks))
             Df Sum Sq Mean Sq F value Pr(>F)
Discipline   13 101139    7780    10.2 <2e-16 ***
Residuals   380 288565     759
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In many texts using R, this assignment and its less often used but equally useful `->` assignment are used in all situations where the outcome of a command is to be stored as an object for further manipulation. The purists may frown on the use of the single `=` character being used to assign something to an object but it's quick, it works and it's understood by the novice.

Utts and Heckard present an interval plot on page 235 of this section. R does not have a command to call on that creates a graph like that given by Minitab.

Example 8.9 uses the `PaperPlanes` dataset. Obtain it using:

```
> data(PaperPlanes, package = "MindOnStats")
> dim(PaperPlanes)
```

```
[1] 144  10
> head(PaperPlanes)
  PlaneNum  Design FlightNum Paper  Thrower FlightTime Displacement
1        1 generic         1  rice    Sarah      0.980         3.13
2        1 generic         2  rice Rachelle      2.132         4.96
3        1 generic         3  rice   Alicia      1.421         3.86
4        1 generic         4  rice     Jake      1.467         6.72
5        2 generic         1 plain   Alicia      1.698         4.93
6        2 generic         2 plain    Sarah      1.024         6.28
  LandPosition  Landing Interference
1         down nosedive           no
2         down nosedive           no
3         down    glide           no
4         down    glide           no
5         down nosedive           no
6         down    glide           no
```

We need only the data for the 'stingray glider' design of paper planes for this example. Sometimes it proves easier to create a new dataset rather than use the `subset` argument because this argument is not available in all commands we might want to apply.

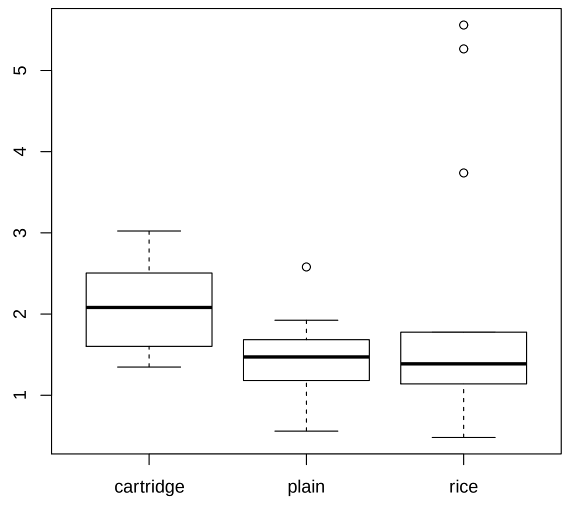
```
> StingRay = PaperPlanes[PaperPlanes$Design == "stingray glider",
       ]
> dim(StingRay)
[1] 48 10
> head(StingRay)
   PlaneNum          Design FlightNum Paper  Thrower FlightTime
49       13 stingray glider         1  rice   Alicia      0.969
50       13 stingray glider         2  rice    Sarah      1.680
51       13 stingray glider         3  rice Rachelle      0.480
52       13 stingray glider         4  rice     Jake      1.287
53       14 stingray glider         1 plain   Alicia      1.680
54       14 stingray glider         2 plain    Sarah      2.580
   Displacement LandPosition  Landing Interference
49         4.44           up nosedive           no
50         4.93           up nosedive           no
51         3.32         down nosedive           no
52         6.09           up    glide           no
53         7.40           up    glide    furniture
54        10.94           up    glide           no
```

We've used the `dim()` and `head()` commands here to show the fact that the irrelevant rows have been removed.

We can investigate the differences among the performances of the paper planes using a boxplot before fitting the one-way ANOVA model.

Figure 8.4 on page 237 of Utts and Heckard.

```
> boxplot(FlightTime ~ Paper, data = StingRay)
```



```
> summary(StingRay.aov1 <- aov(FlightTime ~ Paper, data = StingRay))
            Df Sum Sq Mean Sq F value Pr(>F)
Paper        2    4.1    2.05    2.16   0.13
Residuals   45   42.8    0.95
```

8.3 Assumptions and diagnostics for ANOVA

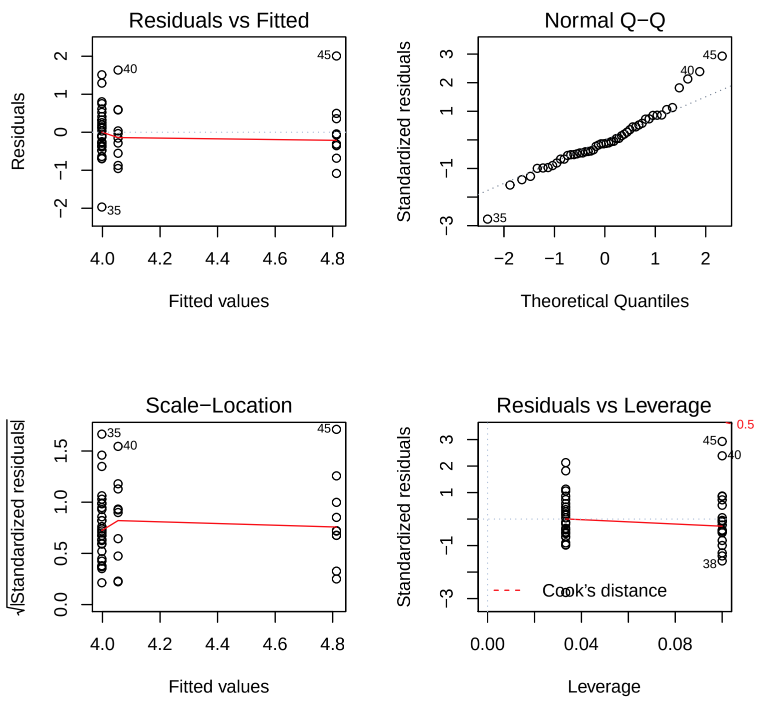
We created (and therefore stored) the model object called `GreenLights.aov` in Section 8.2. We can now investigate the assumptions of the model using the residual analysis graphs found using the `plot()`, `qqnorm()` and `qqline()` commands.

Notice that R generates a set of four plots when the `plot()` command is applied to the model object, as shown in Extra Figure 8.3 on page 54 of this manual. The two graphs we want are then created manually using separate code in Figures **??** and 8.7. The other two graphs are not required for use with Utts and Heckard but may be explained in higher level statistics courses. Use of the `par()` command puts the four graphs into one window in R. The user will need to click through the various windows if this command is not issued. In that case, the user has the option of using one command and then finding the graphs wanted, or issuing the commands that give only the desired graphs.

Notice how in the continuation of Example 8.9 on page 241 we take the logarithm of the response data. This is achieved using the `log()` command inside the single command that does all the work!

Extra Figure 2 is not a replication of work given in Utts and Heckard.

```
> par(mfrow = c(2, 2))
> plot(GreenLights.aov)
```



```
> summary(StingRay.aov2 <- aov(log(FlightTime) ~ Paper, data = StingRay))
            Df Sum Sq Mean Sq F value Pr(>F)
Paper        2   1.41   0.704    3.37  0.043 *
Residuals   45   9.40   0.209
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The residuals for the model using the raw response data are investigated in Figure 8.9, while the logged response data leads to residuals plotted in Figure 8.10.

8.4 Other methods for investigating effects of a categorical variable on a continuous variable

We apply the *Kruskal-Wallis test* for the `PaperPlanes` dataset using the `kruskal.test()`:

```
> kruskal.test(FlightTime ~ Paper, data = StingRay)
        Kruskal-Wallis rank sum test

data:  FlightTime by Paper
Kruskal-Wallis chi-squared = 8.771, df = 2, p-value = 0.01246
```

Figure 8.6 and Figure 8.7, both on page 240 of Utts and Heckard.

```
> Residuals = residuals(GreenLights.aov)
> Fits = fitted(GreenLights.aov)
> plot(Residuals ~ Fits)


> qqnorm(Residuals)
> qqline(Residuals)
```

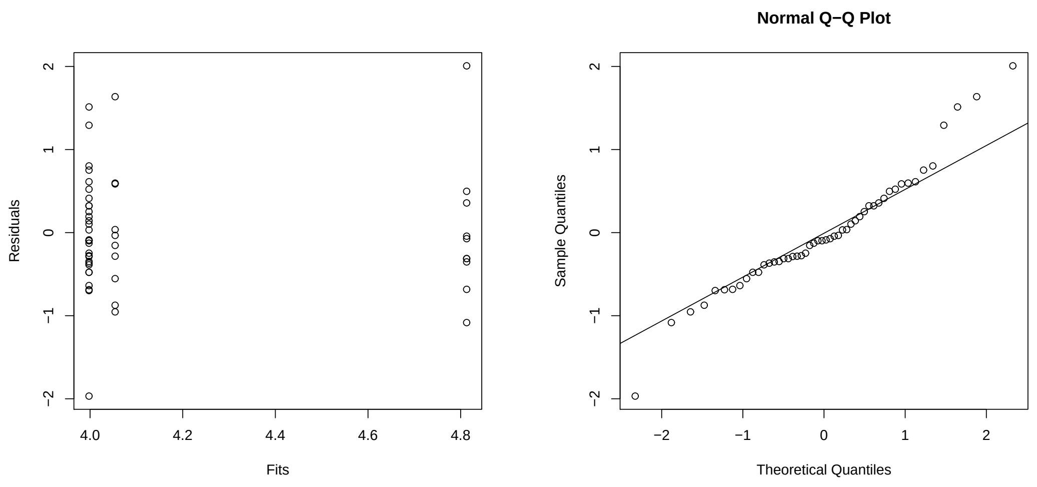


Figure 8.9 on page 241 of Utts and Heckard.

```
> Residuals = residuals(StingRay.aov2)
> qqnorm(Residuals)
> qqline(Residuals)


> Fits = fitted(StingRay.aov2)
> plot(Residuals ~ Fits)
```

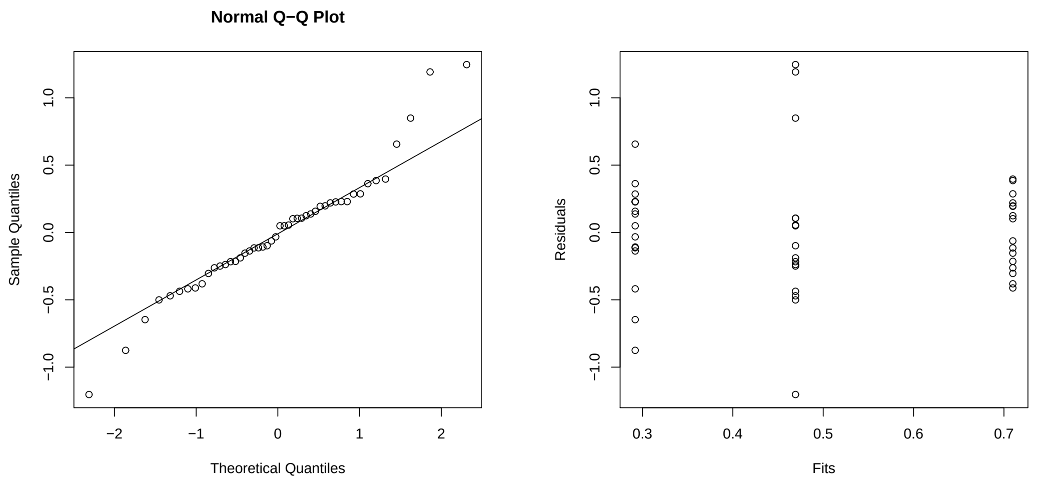
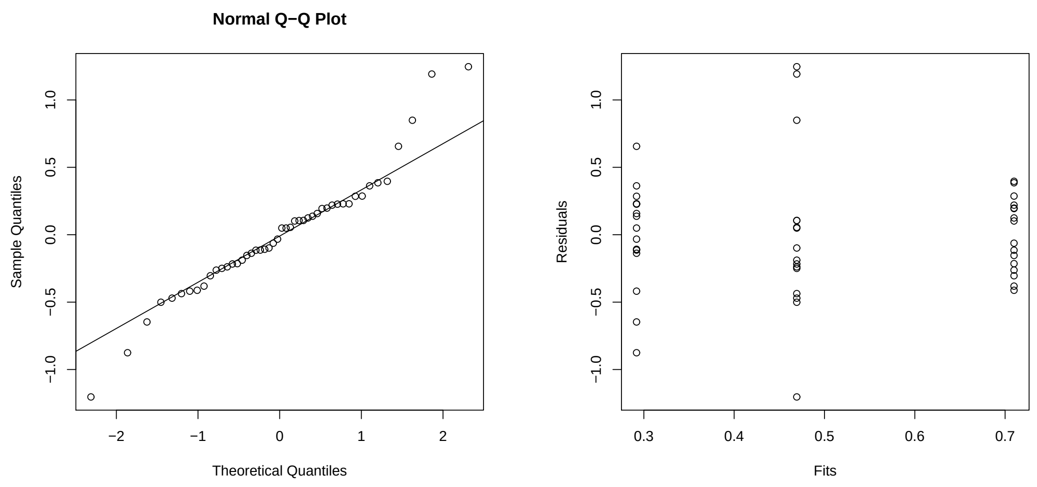


Figure 8.10 on page 242 of Utts and Heckard.

```
> Residuals = residuals(StingRay.aov2)
> qqnorm(Residuals)
> qqline(Residuals)


> Fits = fitted(StingRay.aov2)
> plot(Residuals ~ Fits)
```



Note that Mood's median test is not included in R.

8.5 Multiple comparisons

The set of simultaneous confidence intervals for the differences between all pairs of treatments can be found using the `TukeyHSD()` command. The results from this command applied to the `GreenLights` data can be provided in two ways; we can have the output in text form, but can also use the associated `plot()` command to give us a graphical summary of the comparisons (see Figure 8.13).

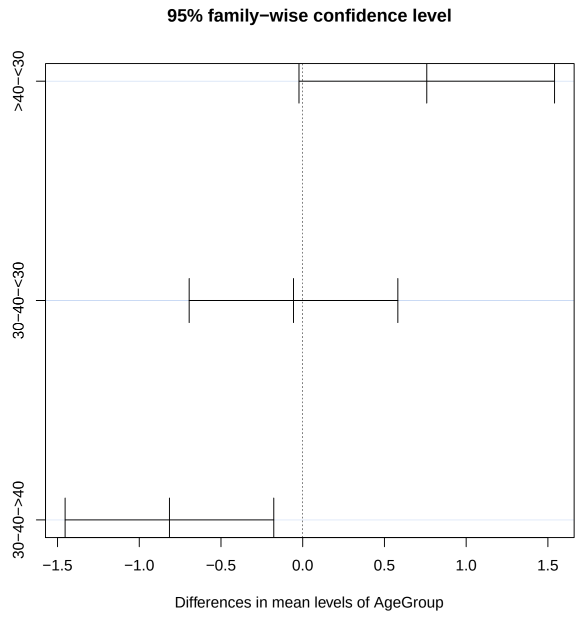
We created (and therefore stored) the model object called `GreenLights.aov` in Section 8.2. We now use this model to show the use of the `TukeyHSD()` command:

```
> GreenLights.HSD = TukeyHSD(GreenLights.aov)
> GreenLights.HSD
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = Time ~ AgeGroup, data = GoGoGo, subset = Lights == "green")

$AgeGroup
              diff      lwr    upr   p adj
```

```
>40-<30    0.75900 -0.02278  1.541 0.0587
30-40-<30 -0.05633 -0.69465  0.582 0.9752
30-40->40 -0.81533 -1.45365 -0.177 0.0092
```

Figure 8.13 on page 248 of Utts and Heckard.

```
> plot(GreenLights.HSD)
```



8.6 Two-way ANOVA

The `aov()` command is used for two-way analyses in a very similar way to the one-way analysis shown above. We just (quite literally) add the second factor to the right-hand-side of the model specified.

We illustrate the two-way analysis of variance model using the `TimePerception` dataset, obtained using:

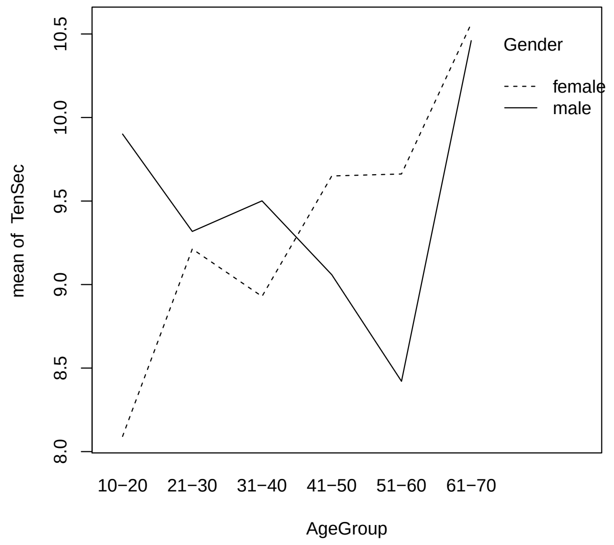
```
> data(TimePerception, package = "MindOnStats")
```

We can see the interaction plot in Figure 8.15, and then fit the model (including an interaction) using:

```
> summary(aov(TenSec ~ Gender * AgeGroup, data = TimePerception))
                 Df Sum Sq Mean Sq F value Pr(>F)
Gender            1    0.3    0.25    0.10  0.749
AgeGroup          5   31.6    6.32    2.55  0.032 *
Gender:AgeGroup   5   27.3    5.46    2.21  0.059 .
Residuals       108  267.4    2.48
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 8.15 on page 251 of Utts and Heckard.

```
> attach(TimePerception)
> interaction.plot(response = TenSec, x.factor = AgeGroup,
       trace.factor = Gender)
> detach(TimePerception)
```



Example 8.10's continuation on page 255 uses a log transformation again.

```
> summary(Planes.aov2 <- aov(FlightTime ~ Paper * Design, data = PaperPlanes))
              Df Sum Sq Mean Sq F value Pr(>F)
Paper          2    0.7    0.35    0.66 0.5167
Design         2    6.8    3.40    6.36 0.0023 **
Paper:Design   4    8.8    2.19    4.10 0.0036 **
Residuals    135   72.2    0.53
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> lnFlightTime = log(PaperPlanes$FlightTime)
> summary(Planes.aov3 <- aov(lnFlightTime ~ Paper * Design,
       data = PaperPlanes))
              Df Sum Sq Mean Sq F value  Pr(>F)
Paper          2   0.03   0.017    0.10 0.90527
Design         2   2.09   1.045    6.03 0.00309 **
Paper:Design   4   4.07   1.017    5.88 0.00022 ***
Residuals    135  23.37   0.173
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The residual analyses for these two models are given in Figures 8.18 (for raw response data) and 8.19 (for logged response data).

Figure 8.18 on page 255 of Utts and Heckard.

```
> Residuals = residuals(Planes.aov2)
> qqnorm(Residuals)
> qqline(Residuals)


> Fits = fitted(Planes.aov2)
> plot(Residuals ~ Fits)
```

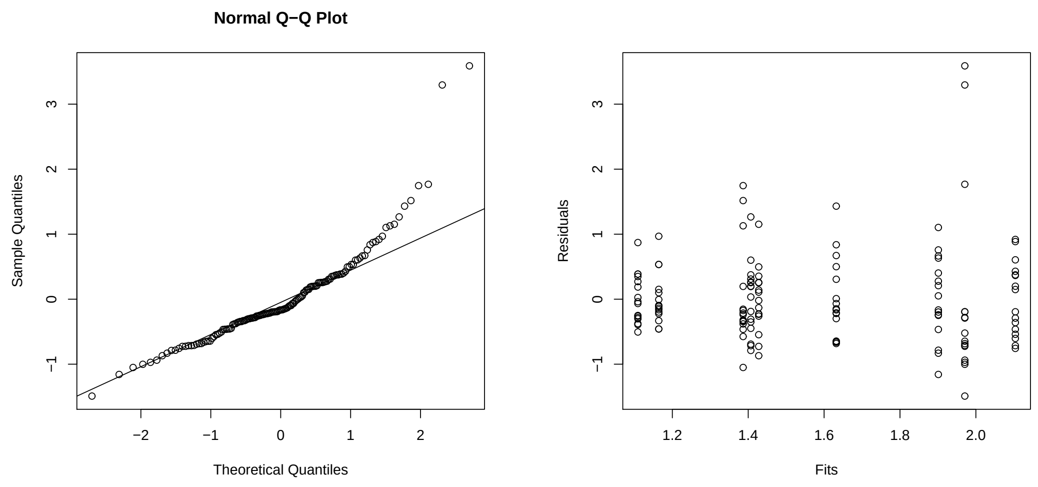
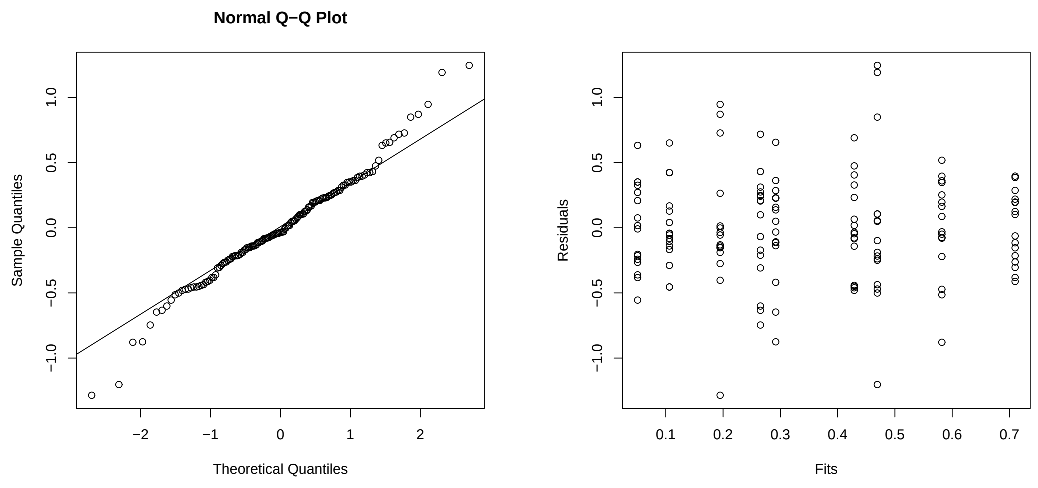


Figure 8.19 on page 256 of Utts and Heckard.

```
> Residuals = residuals(Planes.aov3)
> qqnorm(Residuals)
> qqline(Residuals)


> Fits = fitted(Planes.aov3)
> plot(Residuals ~ Fits)
```



The continuation of Example 8.6 on page 256 where we deal with unbalanced data is important to illustrate. In any situation where new software is used, the user must make themselves aware of the way unbalanced data are handled. We re-order the factors here to show that R uses the sequential sum of squares approach for the `aov()` command.

```
> summary(aov(Time ~ Gender * Lights, data = GoGoGo))
              Df Sum Sq Mean Sq F value Pr(>F)
Gender         1    0.0    0.03    0.07   0.79
Lights         1   13.4   13.39   27.90  8e-07 ***
Gender:Lights  1    0.8    0.84    1.74   0.19
Residuals     96   46.1    0.48
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> summary(aov(Time ~ Lights * Gender, data = GoGoGo))
              Df Sum Sq Mean Sq F value  Pr(>F)
Lights         1   13.4   13.35   27.81 8.2e-07 ***
Gender         1    0.1    0.07    0.15    0.70
Lights:Gender  1    0.8    0.84    1.74    0.19
Residuals     96   46.1    0.48
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

8.7 More on continuous response and categorical explanatory variables

To test for the homogeneity of variance, R has the `bartlett.test()` command for Bartlett's test. Levene's test is not implemented in base R, but is available through a number of additional packages.

Applying Bartlett's test to the `Textbooks` dataset fused in this chapter, we find:

```
> bartlett.test(Price ~ Discipline, data = Textbooks)
        Bartlett test of homogeneity of variances

data:  Price by Discipline
Bartlett's K-squared = 29.6, df = 13, p-value = 0.005383
```

8.8 Models, notation and calculations for ANOVA

There are no examples in this section that require use of R.

Chapter 9

Regression: investigating relationships between continuous variables

9.1 Some examples of data investigations that include continuous responses and quantitative explanatory variables

No examples in this section require use of R.

9.2 Simple linear regression

The `lm()` command fits a linear model to data. It has an associated `summary()` command and we will see later that it also has an associated `plot()` command.

This section of the Asia-Pacific edition of Utts and Heckard relies on the dataset given in the U.S. edition. This dataset (and all others supplied with the U.S. edition) has been included as part of the *MindOnStats* package for your convenience. Obtain the `HandHeight` dataset using:

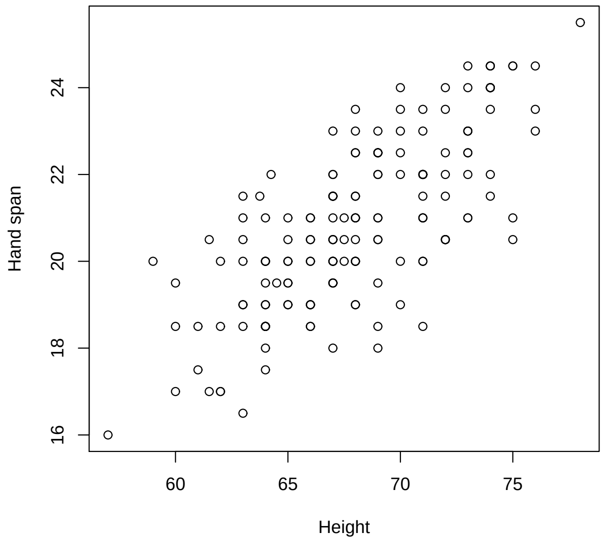
```
> data(HandHeight, package = "MindOnStats")
> str(HandHeight)
'data.frame':	167 obs. of  3 variables:
 $ Sex     : Factor w/ 2 levels "Female","Male": 1 2 2 1 2 1 2 2 1 1 ...
 $ Height  : num  68 71 73 64 68 59 73 75 65 69 ...
 $ HandSpan: num  21.5 23.5 22.5 18 23.5 20 23 24.5 21 20.5 ...
```

We need to wait until the model has been fitted in the next example before illustrating how to add the fitted line to the plot.

We replicate Table 9.1 to provide the R version of the Minitab output in Figure 9.4.

Figure 9.3 on page 294 of Utts and Heckard.

```
> plot(HandSpan ~ Height, data = HandHeight, ylab = "Hand span")
```



```
> summary(Hand.lm <- lm(HandSpan ~ Height, data = HandHeight))
Call:
lm(formula = HandSpan ~ Height, data = HandHeight)

Residuals:
   Min     1Q Median     3Q    Max 
-3.389 -0.935  0.013  1.036  2.663 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  -3.0016     1.6939   -1.77    0.078 .  
Height        0.3506     0.0248   14.11   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 

Residual standard error: 1.3 on 165 degrees of freedom
Multiple R-squared: 0.547,	Adjusted R-squared: 0.544 
F-statistic:  199 on 1 and 165 DF,  p-value: <2e-16 
```

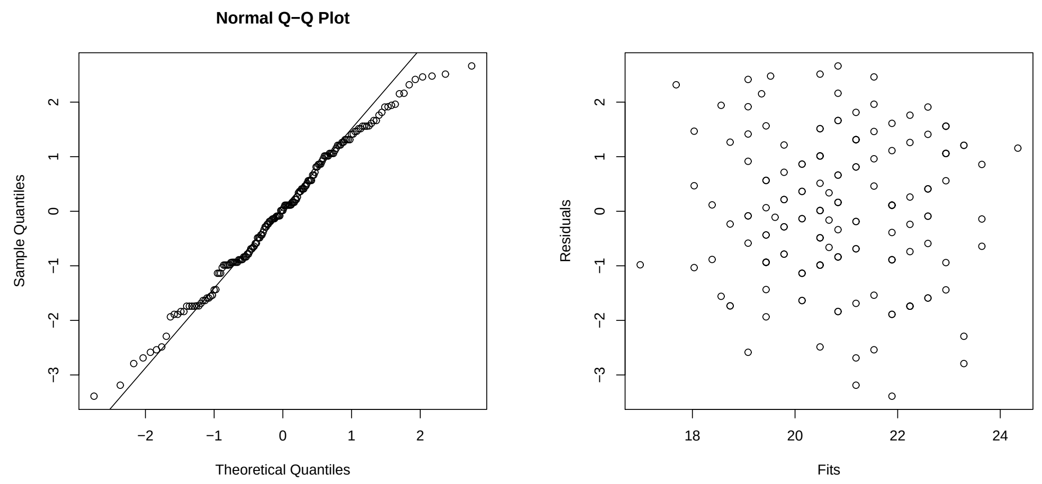
In this example, we introduce the '<-' way of assigning an object to a name. This has the effect of nesting one command inside another in a slightly more elegant way than those shown so far. Different R users will have their preferences over which approach they use. The *residual analysis* for this model is presented in Figure 9.4 and the Extra Figure on page 63 of this manual shows how to add the fitted line to the scatterplot of the data.

To get this line added to the plot we use the `abline()` command. In this case, the command

Figure 9.4 on page 295 of Utts and Heckard.

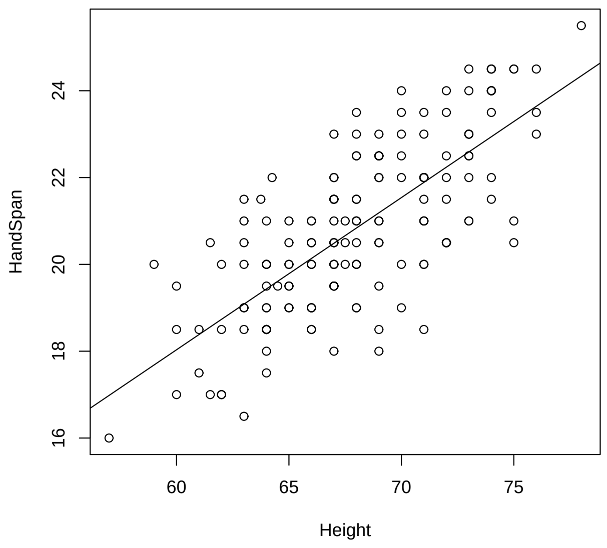
```
> Residuals = residuals(Hand.lm)
> qqnorm(Residuals)
> qqline(Residuals)
```

```
> Fits = fitted(Hand.lm)
> plot(Residuals ~ Fits)
```



Extra Figure 3 is not a replication of work given in Utts and Heckard.

```
> plot(HandSpan ~ Height, data = HandHeight)
> abline(Hand.lm)
```



finds the intercept and slope coefficients from the model and uses them to plot the straight line. We will see later that this command is useful for adding other straight lines to scatterplots.

We need the `Cereals` dataset for several examples in this chapter. Obtain it using the `data()` command as before, and investigate its structure using the `str()` command, then fit the simple regression model using the `summary()` and `lm()` commands as follows:

```
> data(Cereals, package = "MindOnStats")
> str(Cereals)
'data.frame':        43 obs. of  10 variables:
 $ Name        : Factor w/ 42 levels "All Bran","Bran Flakes",..: 1 2 7 8 5 6 4 11 10 21 ...
 $ Manufacturer: Factor w/ 3 levels "Kelloggs","Sanitarium",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ Weight      : int  350 375 340 310 275 400 450 275 300 525 ...
 $ Energy      : int  1399 1428 1602 1626 1582 1681 1607 1670 1637 1489 ...
 $ Protein     : num  15 11.6 6.7 6 7.8 7.1 5.4 6.1 6.4 8.1 ...
 $ Carbo       : num  47.8 59.2 85.6 88.5 83.6 80.8 88.5 86.3 85.5 72.1 ...
 $ Dfibre      : num  27.5 17.4 1.1 1.1 2.6 2.5 1.2 2.3 2.4 8.3 ...
 $ Iron        : num  6.7 10 NA NA 10 10 10 10 10 6.7 ...
 $ Price       : num  2.72 2.88 3.99 3.91 2.28 3.65 4.05 4.06 4.06 4.64 ...
 $ Per100g     : num  0.777 0.768 1.174 1.261 0.829 ...
> summary(Energy.lm <- lm(Energy ~ Dfibre, data = Cereals))
Call:
lm(formula = Energy ~ Dfibre, data = Cereals)

Residuals:
    Min      1Q  Median      3Q     Max
-174.81  -19.50    8.16   38.21  108.36

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1611.67      15.53  103.75  < 2e-16 ***
Dfibre        -10.69       1.25   -8.57  1.1e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 65.8 on 41 degrees of freedom
Multiple R-squared: 0.642,        Adjusted R-squared: 0.633
F-statistic: 73.5 on 1 and 41 DF,  p-value: 1.1e-10
```

We then present the graphs for the residual analysis for this model in Figure 9.6.

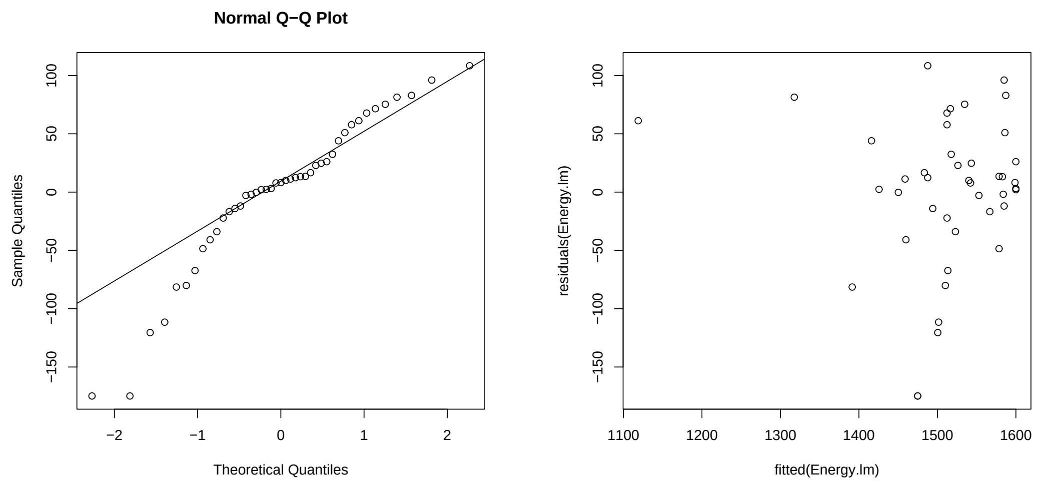
9.3 Messages from residuals

Most examples in this section do not use new ways of working in R.

The example using a quadratic form of the predictor that appears on pages 303-304 cannot be emulated as the data are not available. In this instance, we can either create a new variable

Figure 9.6 on page 299 of Utts and Heckard.

```
> qqnorm(residuals(Energy.lm))
> qqline(residuals(Energy.lm))


> plot(residuals(Energy.lm) ~ fitted(Energy.lm))
```



manually, or use the `poly()` command. We illustrate the use of the `poly()` command on page 71 of this manual.

The graphs that appear in Figure 9.13 use a model created for the textbook data

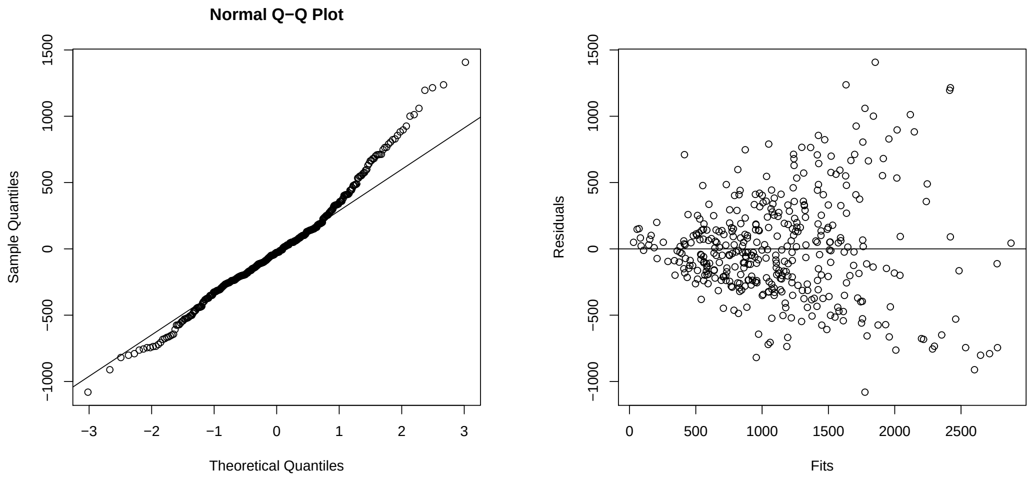
```
> data(Textbooks, package = "MindOnStats")
> summary(BookWeight.lm <- lm(Weight ~ Thickness, data = Textbooks))
Call:
lm(formula = Weight ~ Thickness, data = Textbooks)

Residuals:
    Min      1Q  Median      3Q     Max 
-1080.5  -234.9   -27.3   186.1  1407.1 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)   -54.58      46.88   -1.16     0.25    
Thickness      43.93       1.59   27.70   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 385 on 392 degrees of freedom
Multiple R-squared: 0.662,	Adjusted R-squared: 0.661 
F-statistic:  767 on 1 and 392 DF,  p-value: <2e-16
```

Note that R does not plot a horizontal line on the residuals vs fitted values plot. This can be added manually using the `abline()` command.

Figure 9.13 on page 307 of Utts and Heckard.

```
> Residuals = residuals(BookWeight.lm)
> qqnorm(Residuals)
> qqline(Residuals)
```

```
> Fits = fitted(BookWeight.lm)
> plot(Residuals ~ Fits)
> abline(h = 0)
```



The model can be re-fitted using the transformed response variable; created using the `sqrt()` command:

```
> sqrtWeight = sqrt(Textbooks$Weight)
```

9.4 Multiple regression

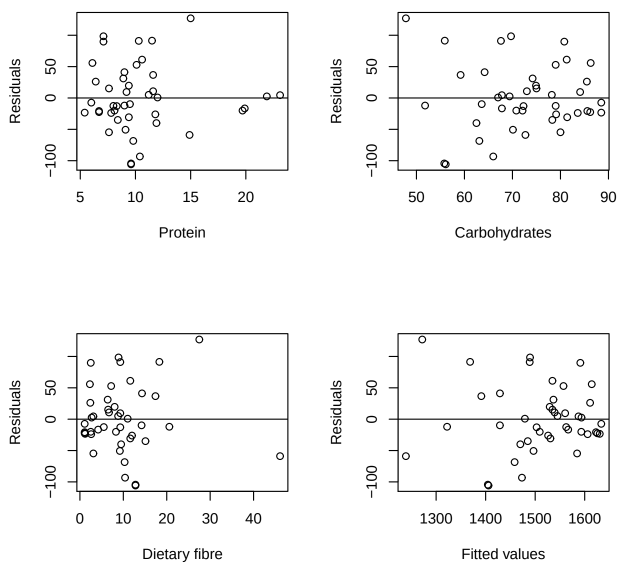
The `lm()` command already introduced is easily extended to allow for multiple predictors in our models. We just add (quite literally) the additional predictors to the right-hand-side of the model.

```
> summary(Energy.lm2 <- lm(Energy ~ Protein + Carbo + Dfibre,
       data = Cereals))
Call:
lm(formula = Energy ~ Protein + Carbo + Dfibre, data = Cereals)

Residuals:
   Min     1Q Median     3Q    Max
-105.8  -25.0   -9.8   28.6  127.0
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1212.83      93.50   12.97  1.0e-15 ***
Protein         3.80       2.27    1.68      0.1
Carbo           4.59       1.04    4.41  8.0e-05 ***
Dfibre         -7.91       1.24   -6.40  1.5e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 55.1 on 39 degrees of freedom
Multiple R-squared: 0.761,       Adjusted R-squared: 0.742
F-statistic: 41.4 on 3 and 39 DF,  p-value: 3.43e-12
```

Figure 9.15 on page 310 of Utts and Heckard.

```
> Residuals = residuals(Energy.lm2)
> Fits = fitted(Energy.lm2)
> par(mfrow = c(2, 2))
> plot(Residuals ~ Cereals$Protein, xlab = "Protein")
> abline(h = 0)
> plot(Residuals ~ Cereals$Carbo, xlab = "Carbohydrates")
> abline(h = 0)
> plot(Residuals ~ Cereals$Dfibre, xlab = "Dietary fibre")
> abline(h = 0)
> plot(Residuals ~ Fits, xlab = "Fitted values")
> abline(h = 0)
```



On page 311 there is an example of different slopes for the two groups. This is actually more easily achieved in R than other statistical sofware such as Minitab.

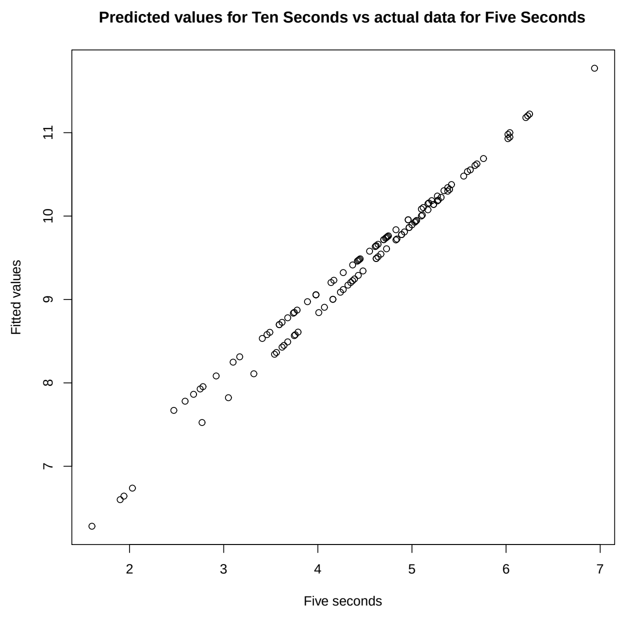
```
> data(TimePerception, package = "MindOnStats")
> summary(TenSec.lm <- lm(TenSec ~ FiveSec * Gender, data = TimePerception))
```

```
Call:
lm(formula = TenSec ~ FiveSec * Gender, data = TimePerception)

Residuals:
   Min     1Q Median     3Q    Max 
-3.790 -0.750 -0.026  0.568  3.703 

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)    
(Intercept)             4.580      0.764    5.99  2.4e-08 ***
FiveSec                 1.063      0.166    6.41  3.3e-09 ***
Gendermale              0.823      1.118    0.74     0.46    
FiveSec:Gendermale     -0.145      0.246   -0.59     0.56    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.34 on 116 degrees of freedom
Multiple R-squared: 0.366,	Adjusted R-squared: 0.349 
F-statistic: 22.3 on 3 and 116 DF,  p-value: 1.8e-11
```

We take the opportunity to show the outcome of the model fitted here by plotting the fitted values against the x-variable. See the Extra Figure 9.4.

Extra Figure 4 is not a replication of work given in Utts and Heckard.

```
> plot(fitted(TenSec.lm) ~ TimePerception$FiveSec, xlab = "Five seconds",
       main = "Predicted values for Ten Seconds vs actual data for Five Seconds",
       ylab = "Fitted values")
```



Note that this is an alternative to plotting the original data using different symbols for the two groups seen in Chapter 3 (page 28 of this manual).

The next example using the `Textbooks` dataset includes both a categorical variable and continuous variables in the model as predictors.

```
> summary(Price.lm1 <- lm(Price ~ Thickness + Weight + Year +
       Coverstyle + Colour + CD, data = Textbooks))
Call:
lm(formula = Price ~ Thickness + Weight + Year + Coverstyle +
    Colour + CD, data = Textbooks)

Residuals:
   Min     1Q Median     3Q    Max
-71.45 -13.81  -0.57  14.24  98.17

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.15e+03   8.60e+02    2.50  0.01301 *
Thickness    6.17e-01   1.81e-01    3.40  0.00074 ***
Weight       2.05e-02   4.01e-03    5.10  5.2e-07 ***
Year        -1.05e+00   4.29e-01   -2.44  0.01519 *
CoverstyleS -3.60e+00   3.15e+00   -1.14  0.25430
ColourY      7.08e+00   3.06e+00    2.31  0.02123 *
CDY         -2.20e+00   4.00e+00   -0.55  0.58334
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 22.7 on 387 degrees of freedom
Multiple R-squared: 0.489,       Adjusted R-squared: 0.481
F-statistic: 61.7 on 6 and 387 DF,  p-value: <2e-16
```

Notice how R has chosen to produce a set of boxplots to depict the differences between the residuals for the categorical variable in Figure 9.16 which plots the residuals against the three predictors. Some statisticians might think to fit the `Year` variable as a factor in this situation; the series of vertically arranged dots for the residuals is common for discrete-valued predictors.

The next model has allowed the continuous predictors to have nonlinear effects on the response:

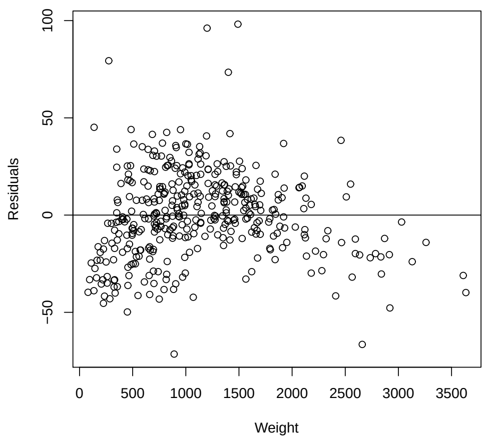
```
> summary(Price.lm2 <- lm(Price ~ poly(Thickness, 2, raw = TRUE) +
       poly(Weight, 2, raw = TRUE) + Year + Coverstyle + Colour +
       CD, data = Textbooks))
Call:
lm(formula = Price ~ poly(Thickness, 2, raw = TRUE) + poly(Weight,
    2, raw = TRUE) + Year + Coverstyle + Colour + CD, data = Textbooks)

Residuals:
   Min     1Q Median     3Q    Max
```

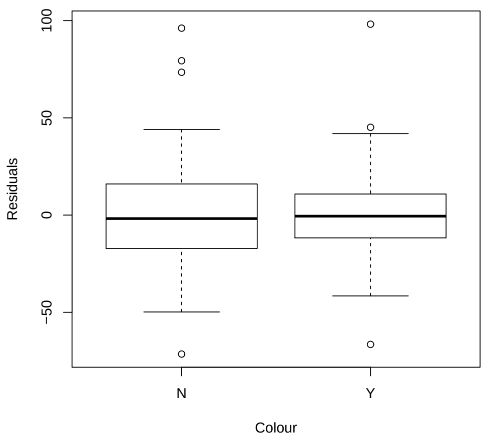
Figure 9.16 on page 312 of Utts and Heckard.

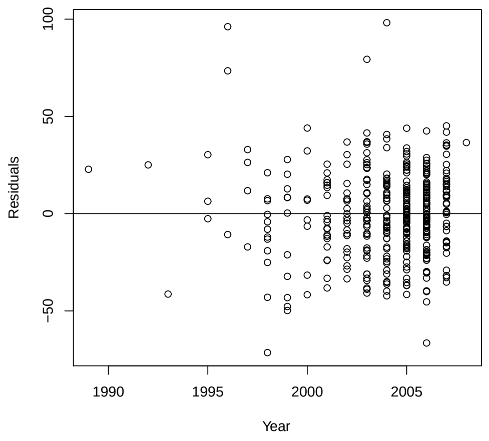
```
> Residuals = residuals(Price.lm1)
> attach(Textbooks)
> plot(Residuals ~ Weight)
> abline(h = 0)
```

```
> plot(Residuals ~ Colour)
```

```
> plot(Residuals ~ Year)
> abline(h = 0)
> detach(Textbooks)
```







```
-77.26 -12.72  -0.11  10.94  90.66

Coefficients:
                                 Estimate Std. Error t value Pr(>|t|)
(Intercept)                      2.30e+03   7.90e+02    2.92   0.0038 **
poly(Thickness, 2, raw = TRUE)1  4.57e-01   4.57e-01    1.00   0.3184
poly(Thickness, 2, raw = TRUE)2 -5.34e-03   6.17e-03   -0.87   0.3870
poly(Weight, 2, raw = TRUE)1     7.32e-02   8.46e-03    8.65  < 2e-16 ***
poly(Weight, 2, raw = TRUE)2    -1.50e-05   2.19e-06   -6.86  2.7e-11 ***
Year                            -1.14e+00   3.94e-01   -2.88   0.0041 **
CoverstyleS                     -3.21e+00   2.89e+00   -1.11   0.2670
ColourY                          1.78e+00   2.85e+00    0.63   0.5317
CDY                             -1.22e+00   3.65e+00   -0.33   0.7390
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 20.7 on 385 degrees of freedom
Multiple R-squared: 0.578,        Adjusted R-squared: 0.57
F-statistic:   66 on 8 and 385 DF,  p-value: <2e-16
```

Note the use of the `poly()` command to create a set of polynomial effects for the `Thickness` and `Weight` variables. The `raw` argument is required; if it is not given explicitly, the variable is centred before the squared term is found.

9.5 Some formulae in regression

Unlike other software, R does not give the full analysis of variance table for the model. Rather we are given just the F-statistic and its hypothesis test. If you want to generate the ANOVA, use the `anova()` command applied to a model object. For example:

```
> anova(Energy.lm)
Analysis of Variance Table

Response: Energy
          Df Sum Sq Mean Sq F value  Pr(>F)
Dfibre     1 317914  317914    73.5 1.1e-10 ***
Residuals 41 177435    4328
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

To find a 95% confidence interval for the slope parameter in our model, we use the `confint()` command. For example:

```
> confint(Energy.lm)
              2.5 %   97.5 %
(Intercept) 1580.30 1643.045
Dfibre       -13.21   -8.173
```

If we wish to make new predictions based on a regression model, we need to create a new data.frame that has the same variables as the original data that was used to create the model. We can then create a *prediction interval* or *confidence interval* for any combination of values for the set of predictors in the model we are working with.

```
> predict(Energy.lm, Cereals[1:5, ], interval = "prediction")
   fit  lwr  upr
1 1318 1176 1459
2 1426 1290 1561
3 1600 1464 1736
4 1600 1464 1736
5 1584 1448 1719
> predict(Energy.lm, Cereals[1:5, ], interval = "confidence")
   fit  lwr  upr
1 1318 1268 1367
2 1426 1397 1454
3 1600 1571 1629
4 1600 1571 1629
5 1584 1557 1611
```

Chapter 10

Interval estimation: one or two continuous variables

10.1 The sample mean

There is no work requiring R in this section.

10.2 Confidence interval for the mean of a continuous variable

To calculate a confidence interval using manual calculations, we will need to obtain quantiles from the t-distribution using the `qt()` command. For example:

```
> qt(c(0.025, 0.975), df = 24)
[1] -2.064  2.064
```

will find the values needed to calculate a 95% confidence interval based on a sample of size 25, which means we have only 24 degrees of freedom. Compare this to the EXCEL tip on page 347.

The `t.test()` command is useful for creating a confidence interval for the population mean when we already have a sample. We use the data given on page 349 to illustrate various examples in this section.

```
> PhoneCalls = c(0.19, 0.78, 0.96, 1.31, 2.78, 3.16, 4.15,
       4.67, 4.85, 6.5, 7.35, 8.01, 8.29, 12.06, 31.75, 32.52,
       33.91, 36.71, 72.89)
```

If the assumptions for the use of the t-distribution approach are not met then we would use the `wilcox.test()` command which calculates the *Wilcoxon signed rank test*.

```
> t.test(PhoneCalls)
```

```
        One Sample t-test

data:  PhoneCalls
t = 3.315, df = 18, p-value = 0.003849
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
  5.26 23.46
sample estimates:
mean of x
    14.36
```

```
> wilcox.test(PhoneCalls, conf.int = TRUE)
        Wilcoxon signed rank test

data:  PhoneCalls
V = 190, p-value = 3.815e-06
alternative hypothesis: true location is not equal to 0
95 percent confidence interval:
  4.15 20.02
sample estimates:
(pseudo)median
         8.058
```

Note that both of these commands generate more output than we need at this time.

At this time there is no functionality within R to perform the Sign Test as described on page 348.

10.3 Difference between two means

There is no work requiring R in this section.

10.4 Confidence interval for difference between two means

The `t.test()` command is useful for creating a confidence interval for the difference between the means of two populations when we already have samples.

We use the `Reflexes` dataset for this example. Obtain it using:

```
> data(Reflexes, package = "MindOnStats")
> str(Reflexes)
'data.frame':        40 obs. of  9 variables:
 $ Gender     : Factor w/ 2 levels "F","M": 1 2 1 1 2 2 1 2 2 2 ...
 $ RightFluoro: num  20.2 21.1 28.2 19.3 26.2 22.9 26.6 19.6 12.1 27 ...
 $ LeftFluoro : num  21.4 25.9 19 28.9 21.6 18.9 27.4 22.3 16.4 15.1 ...
 $ RightClear : num  22.6 19.3 18.1 21 25.5 27.4 27.5 17.4 14.8 22.8 ...
 $ LeftClear  : num  24.9 28.5 25.7 27.8 19.7 18.4 29 21.7 17.9 19.6 ...
 $ Handedness : Factor w/ 2 levels "L","R": 2 2 2 2 1 1 2 2 2 1 ...
 $ Miss       : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
 $ Age        : int  49 53 23 52 54 32 30 22 15 17 ...
 $ AgeGroup   : Factor w/ 3 levels "0-34","35-69",..: 2 2 1 2 2 1 1 1 1 1 ...
```

Utts and Heckard do not show the confidence interval for the paired *t*-test, but this dataset is available and the histograms do appear on page 355. Note the use of the `paired` argument in the `t.test()` command.

```
> attach(Reflexes)
> t.test(RightFluoro, RightClear, paired = TRUE)
        Paired t-test

data:  RightFluoro and RightClear
t = -1.294, df = 33, p-value = 0.2046
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.9729  0.6611
sample estimates:
mean of the differences
                 -1.156
> detach(Reflexes)
```

Turning to data that is not paired, we use the `HoldingBreath` dataset. Obtain it using:

```
> data(HoldingBreath, package = "MindOnStats")
```

We can now see how to use the `var.equal` argument of the `t.test()` command. We also benefit from being able to use the `data` argument, and a model formula to specify the nature of the work we are doing.

```
> t.test(Time ~ Smoker, data = HoldingBreath)
        Welch Two Sample t-test

data:  Time by Smoker
t = -2.86, df = 52.85, p-value = 0.006047
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -19.161  -3.364
sample estimates:
mean in group n mean in group y
          46.34           57.60
> t.test(Time ~ Smoker, data = HoldingBreath, var.equal = TRUE)
        Two Sample t-test

data:  Time by Smoker
```

```
t = -2.411, df = 100, p-value = 0.01775
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -20.532  -1.994
sample estimates:
mean in group n mean in group y
          46.34           57.60
```

Notice that we did not explicitly specify the `var.equal` argument in the first command as the default action is to not assume equal variances.

10.5 Tolerance intervals for individual values

There are no additional R commands to introduce in this section. Basic calculations are required, however.

10.6 Confidence interval for a standard deviation

There is no command in R that will calculate the confidence interval for σ. We illustrate the commands that will assist performing the manual calculations. We use the `qchisq()` command to obtain quantiles from the χ^2-distribution and then follow the necessary steps given on page 366.

```
> qchisq(c(0.025, 0.975), df = 12)
[1]  4.404 23.337
> 12 * 0.024^2/qchisq(c(0.025, 0.975), df = 12)
[1] 0.0015696 0.0002962
> sqrt(12 * 0.024^2/qchisq(c(0.975, 0.025), df = 12))
[1] 0.01721 0.03962
```

Notice the swapping of the order of the two quantities sought by the `qchisq()` command in the last step. This gets the final answer in the correct (more natural) order.

10.7 Sample size required to estimate a mean with a desired precision

This section only uses basic calculations and does not require introduction of any additional R commands.

Chapter 11

Testing hypotheses about means, proportions and variances in one and two samples

11.1 Overview of statistical hypothesis testing

No examples in this section require use of R.

11.2 Testing hypotheses about a proportion

We can use the `prop.test()` command to perform a test on a single proportion. The `binom.test()` is an alternative. See the differences between these commands in the discussion of the related confidence intervals on page 47 of this manual.

In Example 11.7 we need the probability of getting 7 or more correct guesses out of 10 when each question has two options. On page 43 of this manual, we showed how to calculate a set of binomial probabilities. We repeat the task here for this example.

```
> sum(dbinom(7:10, size = 10, prob = 0.5))
[1] 0.1719
> pbinom(6, size = 10, prob = 0.5, lower.tail = FALSE)
[1] 0.1719
```

Then for 8 or more:

```
> sum(dbinom(8:10, size = 10, prob = 0.5))
[1] 0.05469
> pbinom(7, size = 10, prob = 0.5, lower.tail = FALSE)
[1] 0.05469
```

At the bottom of page 387, there is the z-test for a proportion. This is done manually using the `pnorm()` command as follows:

```
> p = 0.5
> n = 100
> s = sqrt(p * (1 - p)/n)
> pnorm(0.6, mean = p, sd = s, lower.tail = FALSE)
[1] 0.02275
> qnorm(0.95, mean = p, sd = s, lower.tail = FALSE)
[1] 0.4178
```

Turning to Example 11.8 on page 390 for another example, we implement the `binom.test()` and `prop.test()` commands after doing manual calculations as follows:

```
> p = 0.5
> n = 203
> s = sqrt(p * (1 - p)/n)
> pnorm(87/203, mean = p, sd = s, lower.tail = FALSE)
[1] 0.9791


> binom.test(87, 203, alternative = "less")
        Exact binomial test

data:  87 and 203
number of successes = 87, number of trials = 203, p-value =
0.02456
alternative hypothesis: true probability of success is less than 0.5
95 percent confidence interval:
 0.0000 0.4887
sample estimates:
probability of success
                0.4286


> prop.test(87, 203, alternative = "less")
        1-sample proportions test with continuity correction

data:  87 out of 203, null probability 0.5
X-squared = 3.862, df = 1, p-value = 0.02469
alternative hypothesis: true p is less than 0.5
95 percent confidence interval:
 0.0000 0.4887
sample estimates:
     p
0.4286
```

```
> prop.test(87, 203, , alternative = "less", correct = FALSE)
        1-sample proportions test without continuity correction

data:  87 out of 203, null probability 0.5
X-squared = 4.143, df = 1, p-value = 0.02091
alternative hypothesis: true p is less than 0.5
95 percent confidence interval:
 0.0000 0.4863
sample estimates:
     p
0.4286
```

Note the need to stipulate the `alternative` argument in both the `binom.test()` and `prop.test()` commands here. The default was two-sided.

Similarly, Example 11.9 on page 392 illustrates another one-sided test:

```
> binom.test(68, 400, p = 0.2, alternative = "less")
        Exact binomial test

data:  68 and 400
number of successes = 68, number of trials = 400, p-value =
0.07323
alternative hypothesis: true probability of success is less than 0.2
95 percent confidence interval:
 0.0000 0.2039
sample estimates:
probability of success
                  0.17


> prop.test(68, 400, p = 0.2, alternative = "less")
        1-sample proportions test with continuity correction

data:  68 out of 400, null probability 0.2
X-squared = 2.066, df = 1, p-value = 0.07529
alternative hypothesis: true p is less than 0.2
95 percent confidence interval:
 0.0000 0.2044
sample estimates:
   p
0.17


> prop.test(68, 400, p = 0.2, alternative = "less", correct = FALSE)
        1-sample proportions test without continuity correction

data:  68 out of 400, null probability 0.2
X-squared = 2.25, df = 1, p-value = 0.06681
```

```
alternative hypothesis: true p is less than 0.2
95 percent confidence interval:
 0.0000 0.2031
sample estimates:
   p
0.17
```

11.3 Testing hypotheses about the difference in two proportions

We can use the `prop.test()` command to test the differences between a pair of proportions. Looking at Example 11.10 on page 395:

```
> prop.test(c(65, 51), c(111, 92))
        2-sample test for equality of proportions with continuity
        correction

data:  c(65, 51) out of c(111, 92)
X-squared = 0.0932, df = 1, p-value = 0.7602
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.1155  0.1780
sample estimates:
prop 1 prop 2
0.5856 0.5543
```

Notice that we enter the two numerators as the first argument and the two denominators as the second argument to the `prop.test()` command here.

11.4 Connection with testing independence in contingency tables

We use the `matrix()` and `c()` commands to enter the data for Example 11.11 from page 396 into a 2×2 matrix.

```
> Ex11.11 = matrix(c(10, 37, 11, 54), nrow = 2)
> Ex11.11
     [,1] [,2]
[1,]   10   11
[2,]   37   54
```

The use of the `prop.test()` command as done previously gives:

```
> prop.test(c(10, 11), c(47, 65))
        2-sample test for equality of proportions with continuity
        correction

data:  c(10, 11) out of c(47, 65)
X-squared = 0.1137, df = 1, p-value = 0.7359
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.1231  0.2102
sample estimates:
prop 1 prop 2
0.2128 0.1692
```

We can use the `chisq.test()` and `fisher.test()` commands to check the independence of the two factors here.

```
> chisq.test(Ex11.11)
        Pearson's Chi-squared test with Yates' continuity correction

data:  Ex11.11
X-squared = 0.1137, df = 1, p-value = 0.7359

> fisher.test(Ex11.11)
        Fisher's Exact Test for Count Data

data:  Ex11.11
p-value = 0.6273
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.4529 3.8329
sample estimates:
odds ratio
     1.323
```

The `chisq.test()` command does not allow for any one-sided hypothesis testing, but the `fisher.test()` command does. In contrast to Minitab (and the output given on page 397), R uses the odds ratio as the basis of the hypothesis test:

```
> fisher.test(Ex11.11, alternative = "greater")
        Fisher's Exact Test for Count Data

data:  Ex11.11
p-value = 0.3655
alternative hypothesis: true odds ratio is greater than 1
95 percent confidence interval:
 0.5302    Inf
sample estimates:
odds ratio
     1.323
```

11.5 Testing hypotheses about one mean

We will use the `t.test()` command to perform a test about a single population mean. In Example 11.12 on page 402, we need the `GoGoGo` dataset again.

```
> data(GoGoGo, package = "MindOnStats")
```

but this time we are only interested in the observations relevant to an amber light. We create another dataset and test that its mean is less than 3.6 seconds by:

```
> AmberLights = GoGoGo$Time[GoGoGo$Lights == "amber"]
> t.test(AmberLights, mu = 3.6, alternative = "less")
        One Sample t-test

data:  AmberLights
t = -1.889, df = 49, p-value = 0.0324
alternative hypothesis: true mean is less than 3.6
95 percent confidence interval:
  -Inf 3.582
sample estimates:
mean of x
    3.441
```

11.6 Testing hypotheses about the mean of paired differences

We will use the `t.test()` command to perform a test about the population mean difference using data from paired samples.

Example 11.13 on page 403 uses data which we enter manually using the `c()` command:

```
> Airport = c(283591, 269620, 312220, 300679, 217889, 381030,
       232288, 186285, 230672, 248172, 221898, 257073)
> City = c(188010, 197874, 193954, 210545, 212116, 277022,
       239715, 197761, 256650, 182655, 146602, 149663)
> t.test(Airport, City, paired = T)
        Paired t-test

data:  Airport and City
t = 3.804, df = 11, p-value = 0.002924
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 24188 90620
sample estimates:
mean of the differences
                  57404
```

The alternative way of working is to create a single sample and use the `t.test()` command as if we were working with a single sample:

```
> Difference = Airport - City
> t.test(Difference)
        One Sample t-test

data:  Difference
t = 3.804, df = 11, p-value = 0.002924
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 24188 90620
sample estimates:
mean of x
    57404
```

11.7 Testing hypotheses about the difference between two means

We will use the `t.test()` command to perform a test about the difference between the means of two populations.

As we do not have the data for the example given, we use the `GoGoGo` dataset again here. The following is not in Utts and Heckard.

```
> t.test(Time ~ Lights, data = GoGoGo)
        Welch Two Sample t-test

data:  Time by Lights
t = -5.277, df = 91.65, p-value = 8.759e-07
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.0059 -0.4557
sample estimates:
mean in group amber mean in group green
              3.441               4.172
```

The only reason for offering this example is that we illustrate the ability to showcase the formula approach to performing the test.

11.8 Non-parametric tests and medians

In Example 11.16 on page 411, we use the `Reflexes` dataset and the `wilcox.test()` to apply the *Wilcoxon signed rank test* to see if there is a difference based on data from paired samples.

```
> data(Reflexes, package = "MindOnStats")
> attach(Reflexes)
> wilcox.test(RightFluoro, RightClear, paired = TRUE, alternative = "less")

        Wilcoxon signed rank test with continuity correction

data:  RightFluoro and RightClear
V = 224.5, p-value = 0.1076
alternative hypothesis: true location shift is less than 0

> detach(Reflexes)
```

Note there are minor differences between the R output and the Minitab tip on page 411.

To show the two-sample version we create an example that does not appear in Utts and Heckard.

```
> wilcox.test(RightFluoro ~ Gender, data = Reflexes)

        Wilcoxon rank sum test with continuity correction

data:  RightFluoro by Gender
W = 220, p-value = 0.4476
alternative hypothesis: true location shift is not equal to 0
```

Note that R does not call this a *Mann-Whitney test* explicitly; reading the help file for the `wilcox.test()` shows that this is in fact what is happening when we apply this command.

11.9 Tests for one or two standard deviations

The Minitab tip on page 415 finds a proportion of values lower than a point on a χ^2-distribution. In R we use the `pchisq()` command. Replicating Example 11.18 on page 415 gives:

```
> TestVal = 28 * 0.25/0.35
> pchisq(20, df = 28)

[1] 0.1355
```

Example 11.19 on page 416 requires use of the F-distribution. The probability required is found using the `pf()` command:

```
> pf(4/2.25, df1 = 9, df2 = 14, lower.tail = FALSE)

[1] 0.1613
```

Note that Utts and Heckard talk about doubling this p-value.

The `var.test()` command does this testing using sample data rather than summarised data as we have in this example.

11.10 The relationship between tests and confidence intervals

No additional R commands need to be introduced in this section.

11.11 t-tests and ANOVA: correspondences and pitfalls of t-testing in real investigations

No additional R commands need to be introduced in this section.

11.12 The rejection region approach to hypothesis testing

No additional R commands need to be introduced in this section.

11.13 Sample size, statistical significance, practical importance and effect size

The `power.t.test()` command is quite powerful in that it will perform the calculations for all situations where the `t.test()` command would be used in the analysis. To replicate the findings of Example 11.24, we show the corresponding code to replace the Minitab tip on page 432 using:

```
> power.t.test(delta = 0.5, sd = 1, sig.level = 0.05, power = 0.8,
       type = "one.sample", alternative = "one.sided")
     One-sample t test power calculation

              n = 26.14
          delta = 0.5
             sd = 1
      sig.level = 0.05
          power = 0.8
    alternative = one.sided
```

The answer provided is non-integer so the need to round up the result is left to the user.

The elements of Table 11.7 on page 431 are the estimated power given the other details. These are found using:

```
> power.t.test(n = 20, delta = 0.8, sd = 1, sig.level = 0.05,
       type = "one.sample", alternative = "one.sided")
```

```
     One-sample t test power calculation

              n = 20
          delta = 0.8
             sd = 1
      sig.level = 0.05
          power = 0.9642
    alternative = one.sided
```

The flexibility of the command is exposed as it does the calculation for the detail that is not specified in the command.

Note that power calculations for proportions are done using the `power.prop.test()` command. It has a similar structure and functionality but is used in situations where the `prop.test()` command will be used for the analysis.

Chapter 12

More on probability, random variables and distributions

12.1 Events

There are no examples in this section that require R.

12.2 Probability rules

There are no examples in this section that require explicit use of R, but some calculations can benefit from knowledge of an additional command.

The `choose()` command finds the number of combinations of size k from a set of size n. Example 12.7 on page 456 uses:

```
> choose(4, 3)
[1] 4
```

12.3 Independence and conditional probability

There are no examples in this section that require R.

12.4 Using conditional probabilities

There are no examples in this section that require R.

12.5 Bayes' theorem

There are no examples in this section that require R.

12.6 Continuous distributions

Example 12.18 uses the uniform distribution. We can extract probabilities from this distribution using the `punif()` command. For example:

```
> punif(4, min = 0, max = 10)
[1] 0.4
```

and we can construct the triangular distribution manually using:

```
> x = 1:10
> Fx = 1 - (10 - x)^2/100
> Fx
 [1] 0.19 0.36 0.51 0.64 0.75 0.84 0.91 0.96 0.99 1.00
```

We can build our own function to do the work for us on any value of x.

```
> Triangular <- function(x) {
       Fx = 1 - (10 - x)^2/100
       return(Fx)
   }
> Triangular(5.4)
[1] 0.7884
```

12.7 A very special process — the Poisson

On page 475 we want $Pr(X > 10)$ where x is Poisson with $\lambda = 12$. We use the `ppois()` command to find such probabilities.

```
> ppois(10, 12, lower.tail = FALSE)
[1] 0.6528
```

For individual probabilities we can use the `dpois()` command. Note the need to set the threshhold at one less than the number we want included as the `ppois()` command is generally adding up the individual probabilities up to and including this point. Use of the `lower.tail` argument finds the complement if set to `FALSE`.

The exponential distribution values needed in the example on page 477 can be found using the `pexp()` command:

```
> pexp(5, rate = 0.4, lower.tail = FALSE)
[1] 0.1353
```

Chapter 13

Sums and differences of random variables

13.1 Examples for which this chapter is needed

There are no examples in this section that require R.

13.2 Sums and differences of two random variables

There are no examples in this section that require R. Note however that the `cor()` command will calculate a correlation coefficient for a pair of samples.

13.3 Means and variances of linear combinations of random variables

There are no examples in this section that require R.

13.4 Linear combinations of normal random variables

There are no examples in this section that require R.

13.5 Sums of some other independent random variables

There are no examples in this section that require R. The interested reader could investigate R's functionality for the gamma distribution and such commands as `pgamma()`.

13.6 The sample mean and the central limit theorem

There are no examples in this section that require R. Simulations proving the central limit theorem might be done using the various random number generating commands for the distributions seen in Utts and Heckard. The interested reader could investigate the `rnorm()`, `rpois()`, `rgamma()`, `runif()`, `rexp()` and `rbinom()` commands.

13.7 Combining 'errors'

There are no examples in this section that require R.

Chapter 14

Some further data situations

14.1 A binary logistic regression

We need the `Reflexes` dataset to replicate Example 14.2 on page 510. Obtain it using:

```
> data(Reflexes, package = "MindOnStats")
```

We fit a binary logistic regression using the `glm()` command. This command has many options and is used for more than just binary logistic regression.

```
> Reflexes.glm = glm(Miss ~ Age, data = Reflexes, family = "binomial")
> summary(Reflexes.glm)
Call:
glm(formula = Miss ~ Age, family = "binomial", data = Reflexes)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.3486  -0.2323  -0.0402  -0.0308   1.8968

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   -9.665      4.806   -2.01    0.044 *
Age            0.118      0.060    1.97    0.048 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 37.098  on 39  degrees of freedom
Residual deviance: 19.726  on 38  degrees of freedom
AIC: 23.73

Number of Fisher Scoring iterations: 8
```

The model statement given here is the most simple possible. The right-hand-side of the model can be made more complicated in the same way as was done for multiple regression in Chapter 9. We can also use the `anova()` and `confint()` commands for the `glm()` command in similar fashion to the work done with the `lm()` command seen in Chapter 9. They are not shown here as they provide information that is beyond the coverage of Utts and Heckard.

Utts and Heckard do present the confidence interval for the Odds Ratio. This is a slightly protracted process in R. First we extract the part of the output given above that is a table containing the two values we need.

```
> Coeff = summary(Reflexes.glm)$coef
> Coeff
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -9.6649    4.80646  -2.011  0.04435
Age           0.1183    0.05997   1.973  0.04848
```

Then (using subscript notation) we find the confidence interval of the log of the Odds Ratio. This corresponds to other manual confidence interval calculations done in this manual.

```
> Coeff[2, 1] + c(-1.96, 1.96) * Coeff[2, 2]
[1] 0.0007903 0.2358849
```

and finally, we convert these values to being the Odds Ratio using the `exp()` command:

```
> exp(Coeff[2, 1] + c(-1.96, 1.96) * Coeff[2, 2])
[1] 1.001 1.266
```

14.2 Failure and survival data

The examples in this section build upon work already demonstrated in previous chapters of this manual. The interested reader who wants to find probabilities for the Weibull distribution can use the `pweibull()` command.

Appendix A

Some useful (but not necessary) additional tasks

A.1 Using the script window

R users do not rely on typing all those commands out at the command line. We usually place the commands for a large job in a text file and use the script window in R to edit and issue some or all of the contents as we see fit.

We recommend looking at the series of files that accompany the *MindOnStats* package which contain all the commands we issued for the creation of this manual, especially those for later chapters where there are quite a few that are logically grouped together. They are all placed in the `RCode` folder within the package.

If you are using the main R console, you will be able to pull down the `File` menu and see options for a `New script` and `Open script`, which will open the script window with a blank or existing file for the respective options.

The beauty of using this window is the ability to copy and paste commands, to run a selection of them after making minor alterations, and being able to save the set of commands at any time for later reference. We recommend experimenting with the script window.

A.2 Installing additional packages

It is common when performing advanced analyses that the base distribution of R doesn't quite cut it. On these occasions we often need an additional package that is not built into the base distribution of R. You'll need an internet connection for this task to work.

In the packages menu you will see the item for installing a package. There are literally thousands of additional packages to choose from, but do not be tempted to waste time looking at packages you don't need.

Search for the *car* package. After you download it, R will install it and if any additional packages were required for it to work, these would have been downloaded as well.

To gain access to the functions and data in a package we will use the `library()` command. For example:

```
> library(car)
```

would load the *car* package if you had been successful when you installed it. Note that you must issue the `library()` command in each R session you need the package; it remains available if you have loaded it in the current session but it is not automatically re-opened even if you save your work on closing R.

Index