

STATISTICAL MODELLING

Appendix A Introduction to R

References: *An Introduction to R*, that is available, along with other manuals, from within the program in the form of PDF files. Other, more general, references are Everitt, B.S. and Hothorn, T. (2006) *A handbook of statistical analyses using R*. Chapman & Hall, London, Verzani, J. (2005) *Using R for introductory statistics*. Chapman & Hall, Boca Raton, Florida, and Maindonald, J. and Braun, J. (2003) *Data analysis and graphics using R*. Cambridge University Press, Cambridge.

| | | |
|------|--|------|
| A.1. | Introduction to R | A-1 |
| A.2. | Some basic concepts | A-2 |
| a) | Performing tasks | A-2 |
| b) | Type of data objects..... | A-3 |
| c) | Naming conventions | A-3 |
| A.3. | An R session | A-4 |
| a) | Initializing Tinn-R and R..... | A-4 |
| b) | Data Entry..... | A-4 |
| c) | Initial graphs | A-5 |
| d) | Statistical analysis..... | A-5 |
| e) | Report generation | A-6 |
| f) | Finishing up | A-6 |
| A.4. | Managing R usage | A-6 |
| a) | Directories, the workspace and objects | A-6 |
| b) | Getting help in R | A-7 |
| c) | Functions in R | A-8 |
| A.5. | Entering data in R..... | A-9 |
| a) | Creating data | A-9 |
| b) | Opening previously stored data | A-10 |
| c) | Import data stored by other programs..... | A-10 |
| A.6. | Manipulating data | A-11 |
| A.7. | Examples..... | A-12 |
| A.8. | Numeric and Character Vectors versus Factors | A-16 |
| A.9 | Exercises | A-19 |

A.1. Introduction to R

R is a free software environment for statistical computing and graphics. It is an implementation of the language S developed at AT&T's Bell Labs. It has a home page at <http://www.r-project.org/> and is downloadable from a network of mirror sites know as the Comprehensive R Archive Network (CRAN) accessible from <http://cran.r-project.org/>.

R provides:

- An extensive and coherent collection of tools for statistical analysis of data;
- Graphical facilities for data display either at a workstation or as hardcopy;

- An effective object-oriented programming language that can be easily extended.

It does not have a graphical user interface and our usage of it will be script-based, which I prefer anyway. This allows me to control the execution and to edit and save as I go so that I can reuse it later. The free editor Tinn-R provides a nice interface to R for editing and executing scripts.

Our version is the Windows version, but there are also Mac and Unix versions. Indeed it was originally developed under Unix.

R capabilities include:

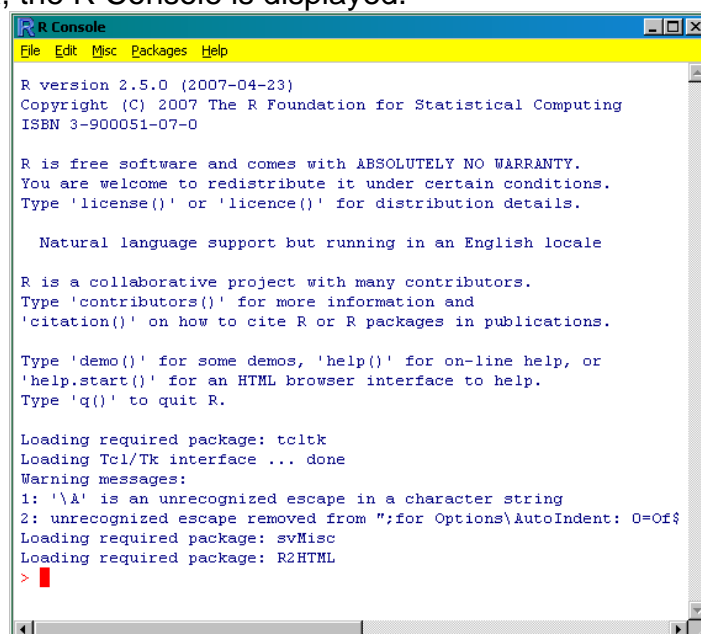
- Import of data from other software packages including Excel spreadsheets
- Summarize data in diagrams with complete control over the detail of the graphs
- Perform calculations on data and basic statistical procedures
- Regression including multiple linear, generalized linear, generalized additive and nonlinear regression
- Design and analysis of experiments
- Linear and nonlinear mixed models
- Multivariate analyses
- Time series analysis

There are also packages written by the R community that extend these capabilities. In particular, I have put together a package called `dae` that provides functions that are useful in the **d**esign and **a**nalysis of **e**xperiments. Instructions for installing it are available on the [Statistical Modelling Resources web site](#).

A.2. Some basic concepts

a) Performing tasks

R is a function-based language. So that tasks are achieved by calling functions. When you start R, the R Console is displayed.



```

R Console
File Edit Misc Packages Help

R version 2.5.0 (2007-04-23)
Copyright (C) 2007 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

Loading required package: tcltk
Loading Tcl/Tk interface ... done
Warning messages:
1: '\A' is an unrecognized escape in a character string
2: unrecognized escape removed from ";for Options\&AutoIndent: 0=Of$
Loading required package: svMisc
Loading required package: R2HTML
>

```

After some messages, there is a prompt “>”. This indicates that R is ready to accept commands. Type `2+2` and see the following:

```
> 2+2
[1] 4
>
```

The answer and another prompt so R is ready for more commands. Type `sqrt(2)` and see:

```
> sqrt(2)
[1] 1.414214
>
```

Suppose I wanted to store some number for later use. The assignment operator “<-” is used. For example,

```
> x <- 1:5
> x
[1] 1 2 3 4 5
>
```

The first statement uses the abbreviation `1:5` to store the sequence of numbers from 1 to 5 in `x`. A prompt is produced. To see what has been stored in `x`, `x` is input on its own.

b) Type of data objects

The entities R operates on are technically known as **objects**.

Vectors: a one-way array of values, all of which are numbers, logical values or character strings, but not a combination of these. Vectors have two attributes, their *length* and *mode*. All values in the array must have the same *mode*; if you supply more than one *mode* they will be coerced to a single *mode*.

Factors: vectors that hold categorical values, like the sex of a person with values “male” and “female”. It has attributes *length* and *levels*. *Levels* are the set of character strings that are allowed in the vector. The *mode* of the *levels* is always character. While you enter the *levels* and operations on factors are in terms of the *levels*, internally the values of a factor are stored as codes.

Matrix: a two-way array of values of the same *mode*. They have the attributes *dim*, *length* and *mode*, with *dim* being the number of rows and columns.

Data frames: a two-way array of values in which columns can be of different modes but must be of the same length.

Lists: A list is an ordered collection of components where each component can be any data object. A list object is the most general and flexible object for holding data in R. They are often used for storing the results of functions that have complex output.

c) Naming conventions

The names of objects can consist of any combination of upper and lowercase letters, numbers (not first) and periods (.) — spaces are not allowed. R is case-sensitive for names. The following are all different, legal names:

Mydata, mydata, data.ozone, RandomNos, lottery.ohio.1.28.90,
data.1, data.2, data.3

In naming your own objects it is better to avoid using system names. While using a system name does not always cause problems, it can and so it is better to not use them.

A.3. An R session

An R session normally consists of:

- a) **Initializing Tinn-R and R:** get Tinn-R and R started
- b) **Data entry:** the data is entered from the keyboard or loaded from a file
- c) **Initial graphs:** produce graphical displays of the data to explore it
- d) **Statistical analysis:** one or more analyses are run on the data
- e) **Report generation:** tables and diagrams that illustrate the results of the analysis are obtained
- f) **Finishing up:** save those files that you want to save

a) Initializing Tinn-R and R

Since Tinn-R is to be used as the interface to R, start it first. As usual for a Windows application, you start Tinn-R by clicking on its icon, either in a desktop folder or in the *Start > Programs* menu. You can then start R by clicking on the *R* button in Tinn-R. When the program starts, it displays the R Console as described above. Associated with an R session is a working directory. This can be checked and changed using *File > Change dir ...* There is also a button in Tinn-R for setting the working directory. On your machine, the default working directory is either the temporary directory for your account or the directory you specified on installation. For example, on my machine it would have been *C:\Documents and Settings\briencj\Local Settings\Temp\Analyses* but I specified *D:\Analyses\R*. In a computer pool, it is *C:\Program Files\R\R-2.5.0* but it is likely that you would want to change it to *E:\My Documents*. If you reset the working directory, it remains in place until you change it again even if there are intervening sessions.

At this point it is a good idea to make sure that you have a suitable working directory. You will also need to create a new file in Tinn-R, for entering a script if you are starting on a new analysis.

b) Data Entry

The first step in virtually any task is to get the data into an R object. A common way of doing this is using the *c* (concatenate) function. The following code sets up a *data frame* object with two columns named *Temp* and *Thrust*:

```
Rocket.dat <-
  data.frame(Temp = c(19,15,35,52,35,33,30,57,49,26),
             Thrust = c(1.2,1.5,1.5,3.3,2.5,2.1,2.5,3.2,2.8,1.5))
```

```

> Rocket.dat
      Temp Thrust
1      19    1.2
2      15    1.5
3      35    1.5
4      52    3.3
5      35    2.5
6      33    2.1
7      30    2.5
8      57    3.2
9      49    2.8
10     26    1.5
>

```

To execute this code it is best to enter it into a file in Tinn-R and then use one of the *Send* buttons on the R toolbar to send it to R for execution. For example, *Send selection* or *Send current line*.

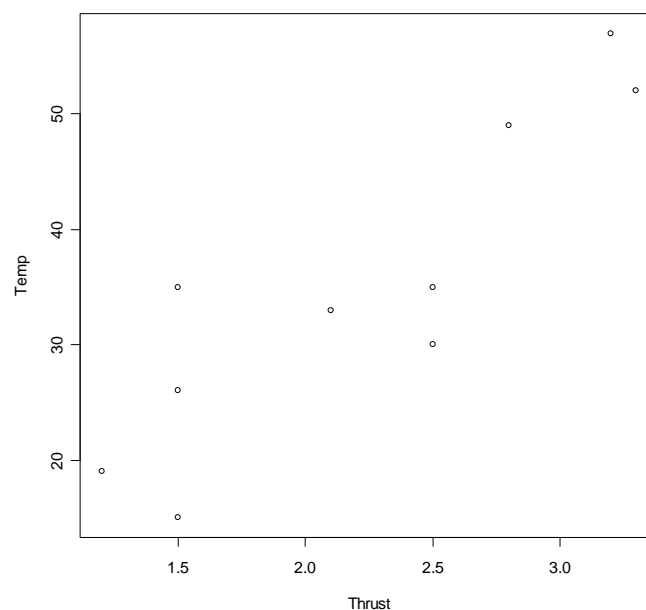
To refer to one of these variable, say `Temp`, one can enter `Rocket.dat$Temp` or `Rocket.dat[["Temp"]]`. To avoid the need to specify the name of the data frame, one can attach the data frame. For example, `attach(Rocket.dat)`.

c) Initial graphs

In this case we may want to obtain a scatter diagram to look at the relationship between `Temp` and `Thrust`. This is simply done using the `plot` function:

```
plot(Thrust, Temp)
```

yielding the following plot:



d) Statistical analysis

In this case we will consider only a simple statistical analysis, the computation of summary statistics using the `summary` function.

```
> summary(Rocket.dat)
      Temp      Thrust
Min.   :15.0   Min.   :1.200
1st Qu.:27.0   1st Qu.:1.500
Median :34.0   Median :2.300
Mean   :35.1   Mean   :2.210
3rd Qu.:45.5   3rd Qu.:2.725
Max.   :57.0   Max.   :3.300
>
```

e) Report generation

In this case, there is little to be done here, except perhaps to print out the graph and statistics. In other cases further graphs and tables of statistics will need to be produced.

A seemingly satisfactory method to incorporate R graphs into Word documents is to use *File > Copy to the clipboard > As a Metafile* in the R Graphics Device window and then pasting it in a Word document. Once in Word it can then be formatted using *Format > Picture* and *Format > Border and Shading* to resize it, add a border and so on.

Copy-and-paste from the Console window can be used to get text output into a Word document.

f) Finishing up

One exits or quits R using the `q()` function. Note must have parentheses; `q` on its own will print out the function `q`, not execute it. Can also click on the `x` in the top right hand corner of the R window.

There will be a message asking whether to save the workspace image. The workspace image contains the objects, data structures and functions, created during a session. Your response to this depends on whether you want to save objects between sessions (see next section). It will be saved into the current working directory with the name `.RData`.

You will also want to save the script file in Tinn-R, for example as *Rocket.r*.

A.4. Managing R usage

a) Directories, the workspace and objects

It is recommended that separate directories are used for different sets of data. Then you will not have to worry about having different names for all the objects and you

can save everything, scripts, graphs, objects and reports, associated with the processing of a data set together in a single folder. This implies that you will have a separate directory for each exercise. For the example I have directory *Rocket*.

Associated with a session is the working directory, which is the default directory for the session.

The *File* menu in the R console and some buttons in Tinn-R can be used to manipulate the working directory, the workspace and objects.

The working directory can be checked and changed using *File > Change dir ...* in the R Console. It can also be set to the current path in Tinn-R using the *Set work directory (current file path)* button or *R > Controlling R > Set work directory (current file path)*.

Workspaces can be loaded and saved from the *File* menu in R. Note that loading a workspace does not clear the current workspace. Rather it adds to the current workspace. So if you do not want objects currently loaded all can be cleared from Tinn-R using the *Clear all objects* button or this command from the *R > Controlling R* menu. The `rm` function in R can be used to delete specific objects.

To find out what objects are in the current workspace use the `ls()` function in the R Console or the *List all objects* button in Tinn-R. For example,

```
> ls()
[1] "MnThrust"    "Rocket.dat"
>
```

So it contains a vector `MnThrust` and a data frame called `Rocket.dat`. Note that `Rocket.dat` is the full name of the data frame and that, in particular, `.dat` is not a file extension.

There are several functions that allow us to find out about objects: `str`, `class`, `attributes`, `names` and `length`. Also, putting in the name of an object will cause its contents to be printed. There are buttons in Tinn-R that *Print content*, *List names* and *List structure* of selected objects, providing a more convenient way of executing these functions.

b) Getting help in R

There are three forms of help for R:

- Online help for functions via the `help` function. For example, `help(log)` provides help on the `log` function. You can also access this from Tinn-R by selecting the function and hitting the F1 key or clicking on the *Help (selected)* button.
- Electronic manuals that can be accessed with *Help > Manuals (in PDF)*. The most important of these is *An introduction to R*.
- Published books such as Maindonald and Braun as well as Venables and Ripley.

c) Functions in R

A **function** is an R expression that returns a value, usually after performing some operation on one or more **arguments**. In a language sense, a function is like a verb and the arguments are the nouns.

As we have been saying, functions are used to achieve tasks in R. For example the plot and calculation of the mean for the Rocket data could be achieved by entering the following commands into the Console window, providing `Rocket.dat` is available in the current workspace:

```
attach(Rocket.dat)
plot(Thrust, Temp)
MnThrust <- mean(Thrust)
MnThrust
```

As I have said I would do this by entering the commands into a file in Tinn-R and sending them to R from there. The output in the Console window from these commands is:

```
> attach(Rocket.dat)
> plot(Thrust, Temp)
> MnThrust <- mean(Thrust)
> MnThrust
[1] 2.21
```

A number of points arise from the use of these commands:

1. As we have already seen, to access the columns of a data frame without specifying the name of the data frame, you must use the `attach` function to attach it — it places it in the Search Path for functions. If you make changes to the data frame subsequent to attaching it, they will not be reflected in the attached data frame. This is because a copy is made for attaching and the changes will not be made to the copy, but to the original. So if you are going to make changes to an attached data frame, then you need to `detach` it before making the changes and `attach` it again after the changes have been made. If you do not `detach` it then you will get a warning message that some columns of the data frame are masked. These are the versions in the previously attached data frame and so the message can be ignored. To find out what is in the search path use `search()`.
2. Functions can be called with or without assignment. For example, the `plot` function is called *without* assignment and its results are displayed, in this case in an R Graphics Device window. The `mean` function is called *with* assignment, the operator `<-` being the assignment operator. The result of the function is assigned to the object `MnThrust`, a vector of length one. No output is produced in this second case. If the function had been called without assignment, the result would have been printed but not stored.
3. Typing in the name of an object results in its being printed. The `[1]` at the start of the line indicates that the number 2.21 is the first element of the object `MnThrust`. Generally, such an index is put at the start of each line of output of

an object so that you can gauge how many elements have been printed on each line and to assist you in finding a particular element.

Also note some more general points about using the Console window:

- Commands are ended by pressing RETURN, but if the statement is not correctly terminated a continuation prompt “+” is used reminding you that more input is necessary. This can be useful when entering long statements.
- The Up and Down arrows can be used to scroll backward and forward through the list of commands. So a typing error can be easily corrected, or a new similar command entered without completely retyping it.
- All commands entered into the Console window are saved in History file that is saved when the workspace is saved and reloaded when the workspace is reloaded.

Some general rules for using functions

R is case-sensitive for names, including those of functions.

Functions may have their arguments **specified** or **unspecified**. There is a concatenation function `c` that puts a list of values into a vector. Its arguments are unspecified. For example

```
y <- c(2.5, 2.7, 2.1)
```

places the three values, supplied as arguments to `c`, in the vector object named `y`.

`plot` has specified arguments where the argument is given in the form `name = value`. When this is the case a mixture of conventions can be used. So for example a plot of the two vectors `Weight` and `Mileage` with a logarithmic y-axis could be obtained using either of the following commands:

```
plot(x=Weight, y=Mileage, log="y")
plot(Weight, Mileage, log="y")
```

A.5. Entering data in R

a) Creating data

We have seen how the `c` function can be used to enter data from the keyboard. However, this function is inconvenient if have the data does not have commas. Another function that requires only spaces is the `scan` function. Input is terminated by a blank line.

```
> Temp=scan()
1: 19 15 35 52 35 33 30 57 49 26
11:
Read 10 items
>
```

If a particular value is missing in the sense that a value was not recorded for it, then enter NA to indicate this. For example, if a respondent answered most, but not all, questions in a questionnaire, an NA would be entered for those questions that were not answered.

In our case, we will often want to enter factors and a response variable into a `data.frame`. A detailed description of setting up a `data.frame` for an experiment is given in Appendix C1, *Entering the results of an experiment into a data.frame*.

b) Opening previously stored data

Data, with all its attributes, can be stored in `.rda` files using the `save` function and then later opened in an R session using the `load` function. For example,

```
save(Rocket.dat, file="Rocket.dat.rda")
load("Rocket.dat.rda")
```

The `.rda` files are binary format files.

c) Import data stored by other programs

The simplest way to do this is to save the data as a comma separated (`csv`) file in the other program, Excel, S-Plus and so on. The `read.table` can be used to import the data. Similarly `write.table` can be used to export data to other programs. The following command reads in a file:

```
CRDRat.dat <- read.table("CRDRat.dat.csv",
                        header= TRUE, sep="," ,
                        colClasses = c("factor", "factor", "numeric"))
```

The following writes a file that can be imported to Excel:

```
write.table(CRDRat.dat, file = "CRDRat.dat.csv",
            sep="," , col.names=NA)
```

The package `foreign` provides commands to import data from specific packages. For example `read.S` imports data from S-Plus.

A.6. Manipulating data

A very important way of manipulating vectors and data frames is through indices. They enable one to subset and reorder data. The following commands show this for the `Rocket.dat` data frame. Note that a data frame is an object with two indices, the first for rows and the second for columns, whereas a vector has only one index. The indices are placed between square brackets. They can be numbers or, if available, names.

```
> attach(Rocket.dat)
> T <- Temp[1:3]
> T
[1] 19 15 35
> Th <- Rocket.dat[, "Thrust"]
> Th
[1] 1.2 1.5 1.5 3.3 2.5 2.1 2.5 3.2 2.8 1.5
> class(Th)
[1] "numeric"
> Th <- Th[order(Th)]
> Th
[1] 1.2 1.5 1.5 1.5 2.1 2.5 2.5 2.8 3.2 3.3
>
```

There are many functions that can be applied to an object. There are those that return a single value, like `min`, `max`, `sum` and `mean`, and those that return vectors, like `sin`, `cos`, `log` and `sqrt`. You can have a combination like

```
Deviation <- Thrust - mean(Thrust)
```

That takes the mean of Thrust off each value.

A particularly useful function for generating patterned data is the `rep` function.

```
> rep(c(2,3,5), each = 3, times = 2)
yields
2 2 2 3 3 3 5 5 5 2 2 2 3 3 3 5 5 5
```

You may also need to change the class of an object. For this there are many `as.something` functions that coerce the argument to class something. For instance `as.factor` converts a numeric to a factor:

```
> A <- rep(1:3, each = 3, times = 2)
> class(A)
[1] "numeric"
> A <- as.factor(A)
> class(A)
[1] "factor"
>
```

A.7. Examples

Example I.1 House price (continued)

For this example, involving the response variable Price and the explanatory variables Age and Area, the data was as follows:

| Age years (x_1) | Area 000 feet ² (x_2) | Price \$000 (y) |
|---------------------------|--|---------------------------|
| 1 | 1 | 50 |
| 5 | 1 | 40 |
| 5 | 2 | 52 |
| 10 | 2 | 47 |
| 20 | 3 | 65 |

One must first enter this data into R. I have already done this and saved it in a file called *House.Prices.dat.rda*.

A set of R functions to obtain the regression analysis is as follows:

```
#Set working directory and load House.Prices.dat
load("House.Prices.dat.rda")
attach(House.Prices.dat)
pairs(House.Prices.dat, pch=16)
#Copy scatterplot matrix to clipboard

House.lm <- lm(Price ~ Age + Years, House.Prices.dat)
summary(House.lm)
residuals <- residuals(House.lm)
fitted <- fitted(House.lm)
windows() #opens new graphics window
plot(fitted, residuals, pch=16)
#Copy residuals-versus-fitted-values plot to clipboard

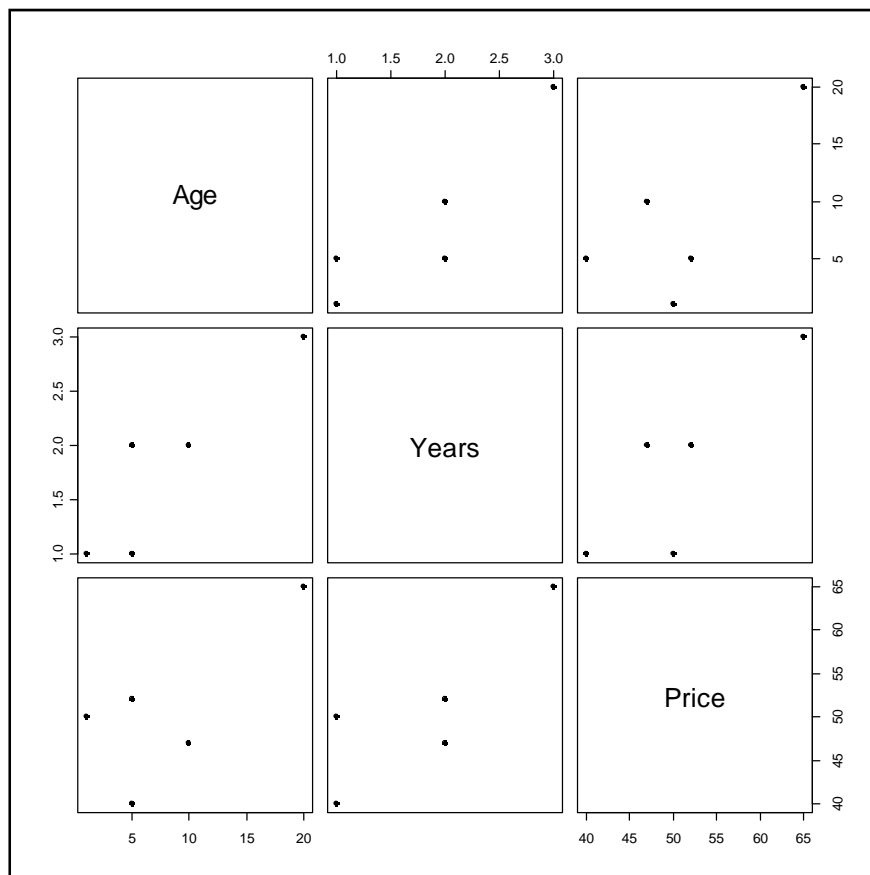
qqnorm(residuals, pch=16)
qqline(residuals)
#Copy qq plot to clipboard

detach(House.Prices.dat)
House.Prices.dat
data.frame(House.Prices.dat, residuals, fitted)
remove("residuals", "fitted")
```

Note the use of the `windows` function (do not forget the “()” or it will just print out the function i.e. what the object `windows` stores) to open a new graphics window so that the previous plot is not overwritten.

R output

The graphs and text output produced by these commands is given below.



```
> House.lm <- lm(Price ~ Age + Years, House.Prices.dat)
> summary(House.lm)
```

```
Call:
lm(formula = Price ~ Age + Years, data = House.Prices.dat)
```

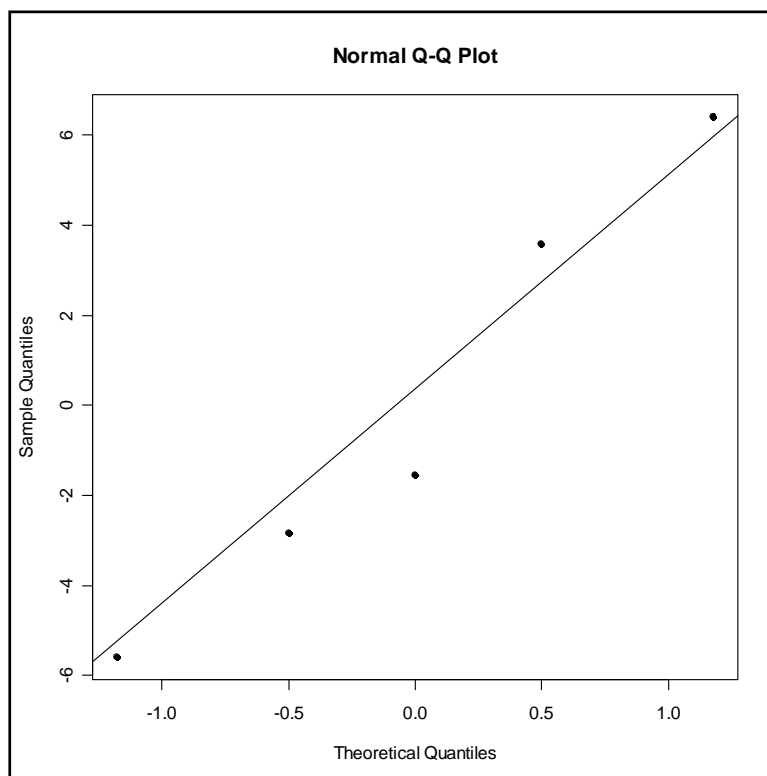
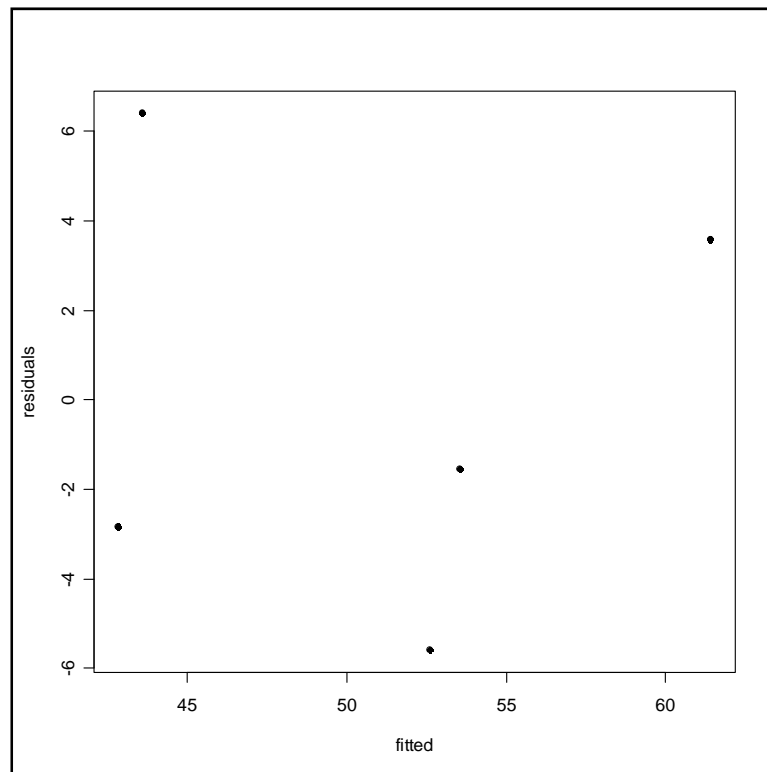
```
Residuals:
    1      2      3      4      5 
6.409 -2.832 -1.551 -5.602  3.576
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  33.0626    10.5066   3.147  0.0879
Age          -0.1897     1.1106  -0.171  0.8801
Years         10.7182     9.7272   1.102  0.3854
```

```
Residual standard error: 6.916 on 2 degrees of freedom
Multiple R-Squared:  0.7142,    Adjusted R-squared:  0.4285 
F-statistic: 2.499 on 2 and 2 DF,  p-value: 0.2858
```

From this output we obtain the fitted equation:

$$\text{Price} = 33.0626 - 0.1897 \text{ Age} + 10.7182 \text{ Years}$$



Example III.1 Rat experiment

Consider an experiment in which the effects of three diets on rats is to be investigated. Suppose I have 6 rats to be fed one of three diets, 3 rats to be fed diet A, 2 diet B and 1 diet C. Then, at the end of the experiment, the experimenter

measured the liver weight as a percentage of total body weight. The results of the experiment, in standard order, are as follows:

| Rat | 5 | 6 | 1 | 3 | 2 | 4 |
|-----------|-----|-----|-----|-----|-----|-----|
| Diet | A | A | A | B | B | C |
| Liver wt. | 3.3 | 3.1 | 2.9 | 3.2 | 3.4 | 2.7 |

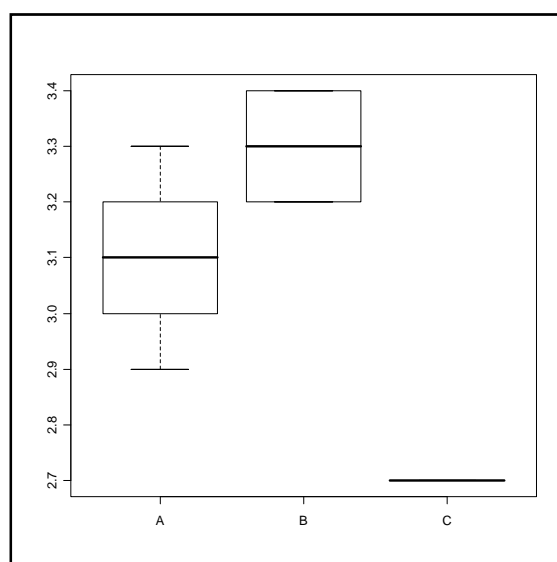
A set of R functions to analyse this data is as follows:

```
load("CRDRat.dat.rda")
attach(CRDRat.dat)
boxplot(split(LiverWt, Diet))
#
# AOV with Error
#
Rat.aov <- aov(LiverWt ~ Diet + Error(Rat), CRDRat.dat)
summary(Rat.aov)
model.tables(Rat.aov, type = "means")
```

Firstly, note that the `split` function in the `boxplot` function splits `LiverWt` into groups defined by `Diet`. Secondly, the use of the `aov` function without the `Error` function will produce a traditional aov table, whereas the `aov` function with the `Error` function will produce the table that displays the confounding in the experiment.

The output produced by the above R expressions is as follows:

```
> load("CRDRat.dat.rda")
> attach(CRDRat.dat)
> boxplot(split(LiverWt, Diet))
```



```
> #
> # AOV with Error
> #
```

```
> Rat.aov <- aov(LiverWt ~ Diet + Error(Rat), CRDRat.dat)
> summary(Rat.aov)
```

```
Error: Rat
```

```
      Df    Sum Sq   Mean Sq F value Pr(>F)
Diet    2  0.240000  0.120000    3.6 0.1595
Residuals 3  0.100000  0.033333
```

```
> model.tables(Rat.aov, type="means")
```

```
Tables of means
```

```
Grand mean
```

```
3.1
```

```
Diet
```

```
      A    B    C
      3.1  3.3  2.7
rep  3.0  2.0  1.0
```

■

A.8. Numeric and Character Vectors versus Factors

Consider the following set of data that is a set of mileages and types of cars from a small study.

| Mileage | Type |
|---------|---------|
| 33 | Small |
| 33 | Small |
| 37 | Small |
| 20 | Sporty |
| 27 | Sporty |
| 19 | Sporty |
| 27 | Compact |
| 23 | Compact |
| 26 | Compact |

Notice that the mileages are positive integers and that the types of car are words. An issue to be considered is what sort of data objects do we want to end up with — numeric vectors, character vectors, factors? The mileages are straightforward — they would go into a numeric vector as they are arbitrary numbers. However, the type of car is different as it has only three values and they are words. We could:

1. decide on some codes — 1 = small, 2 = sporty and 3 = compact — and enter these into a numeric vector;
2. enter the words into a Character vector; or
3. enter either codes or words into a factor.

The following diagram shows the Type data entered into columns of a data frame with a column for each of these data-object types — the order is the reverse of that

listed above. Note that row names have been specified, using the `rownames` function, for this data frame.

```
> TestData.dat
      Mileage      Type Type.code Type.char Type.num
Eagle Summit 4      33    Small      1     Small      1
Ford Escort 4      33    Small      1     Small      1
Ford Festiva 4      37    Small      1     Small      1
Chevrolet Camaro V8 20   Sporty      2     Sporty      2
Dodge Daytona      27   Sporty      2     Sporty      2
Ford Mustang V8     19   Sporty      2     Sporty      2
Audi 80 4          27   Compact      3     Compact      3
Buick Skylark 4     23   Compact      3     Compact      3
Chevrolet Beretta 4 26   Compact      3     Compact      3
> str(TestData.dat)
'data.frame':   9 obs. of  5 variables:
 $ Mileage : num  33 33 37 20 27 19 27 23 26
 $ Type    : Factor w/ 3 levels "Compact","Small",...: 2 2 2 3 3 3 3 1 1 1
 $ Type.code: Factor w/ 3 levels "1","2","3": 1 1 1 2 2 2 3 3 3
 $ Type.char: chr  "Small" "Small" "Small" "Sporty" ...
 $ Type.num : num  1 1 1 2 2 2 3 3 3
```

Factors treat the values as categories and are treated specially in analyses by R. So the decision is important. It is most likely that the type of car would be entered into a factor and it is probably more informative to enter the words. While entering the words is more work, it is possible to enter numeric codes when creating a factor and use the `labels` option to add words. Otherwise you can enter the numeric codes and use the `levels` function on the left side of an assignment to change the levels to words:

```
> Types <- TestData.dat$Type.code
> Types
[1] 1 1 1 2 2 3 3 3
Levels: 1 2 3
> levels(Types) <- c("Small","Sporty","Compact")
> Types
[1] Small Small Small Sporty Sporty Sporty Compact Compact Compact
Levels: Small Sporty Compact
```

Factor data structures

A factor is a vector that has only a limited set of possible values. The possible values are called the levels of the factor. If t is the number of levels of a factor and n is the number of values recorded for the factor, R stores n codes each between 1 and t , rather than the actual levels. It also keeps a list of the t levels associated with the t codes. An error will occur if you attempt to enter a value or code outside those specified.

Because factors involve only a limited set of values, each value often occurs repeatedly. Also, there is often a regular pattern to the values, such as is illustrated in the following values for three factors A, B and C:

```
A: 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
B: 4 4 4 4 1 1 1 1 2 2 2 2 4 4 4 4 1 1 1 1 2 2 2
C: 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
```

Because of this it is often convenient to use the `rep`, `factor` and `data.frame`. The following functions will create a `data.frame` containing the three factors.

```
A <- factor(rep(c(1,2), each=12))
B <- factor(rep(c(4,1,2), each=4, times=2))
C <- factor(rep(1:4, times=6))
Fac3.ran <- data.frame(A,B,C)
```

The first argument of the `rep` function gives the levels of the `factor` and the `each` argument is used to repeat each value a specified number of times while the `times` argument is used to repeat the whole pattern specified by the first argument and `each`.

However, these factors, with patterned values, can be generated using `fac.gen`, available in the `dae` package. It has the following arguments:

```
fac.gen(generate, each=1, times=1, order="standard")
```

The argument `generate` is a list named objects and numbers that specify the factors whose levels are to be generated and the pattern in these levels. If a component of the list is named, then the component should be either a) a single numeric value that is the number of levels, b) a numeric vector that contains the levels of the factor, or c) a character vector that contains the labels of the levels of the factor. The `order="standard"` specifies that the values should be generated with the first factor changing slowest and the last changing fastest. The other possibility is to use `Yates order` in which the speed of change is reversed to that for standard order. So the following expressions will generate the factors:

```
Fac3.ran <- fac.gen(list(A = 2, B = c(4,1,2), C = 1:4))
```

Ordered factors

Sometimes the order of the factors is arbitrary and R orders the factors alphabetically. For example, the factor `Sex` with levels `male` and `female` would be ordered with `female` assigned code 1 and `male` assigned code 2. However, sometimes there is an innate order to the levels that is different to the alphabetical order. For example, a factor for income with values `lo`, `mid` and `hi` would by default have the levels ordered `hi`, `lo`, `mid`. One can specify the order of the levels by converting the factor object to be an ordered object. An ordered object is the same as a factor except that the levels are specifically ordered.

For example to convert the already existing income factor object to an ordered object, use the `ordered` function as follows:

```
ordered(income) <- c("lo", "mid", "hi")
```

Ordered factors are in some circumstances treated differently by R.

A.9 Exercises

Exercises A.2 to A.5 below were covered in text above. Exercise A.1 is based on material given in Maindonald and Braun (2003, chapter 1) *Data analysis and graphics using R*. Cambridge University Press, Cambridge.

A.1. R graphics

One of R's strengths is its graphics. To see some of the potential enter `demo(graphics)`. Press enter after each graph.

Basic plots

The basic command is the `plot` function. Try the following plots, entering the commands into a Tinn-R script file and send them to R from there. (Note that `sin` function expects angles in radians and angles in degrees are converted to radians by multiplying them by $\pi/180$.)

```
plot((0:20)*pi/10, sin((0:20)*pi/10))
plot((1:50)*0.92, sin((1:50)*0.92))
```

Seeing the pattern in the second plot is difficult and one might mistake it for a random pattern. To bring out the pattern reduce the vertical size of the Graphics Device window. (Place the mouse over the bottom border and, when it turns into a double-ended arrow, hold the left mouse button down and drag the bottom border upwards.)

Several plots per graph

If you want more than one plot on a graph, use the `mfrow` argument of the `par` function. For example enter the following commands into a Tinn-R script file and send them to R.

```
oldpar <- par(mfrow=c(3,1))
plot((0:20)*pi/10, sin((0:20)*pi/10))
plot((1:50)*0.92, sin((1:50)*0.92))
par(oldpar)
```

The first statement, as well as setting the `mfrow` argument, also saves the settings prior to doing that in `oldpar`. The settings are then restored to the old settings in the final statement.

Line plots

Use the following statements to create a joined plot of the ACT population 1917–77. Note the use of the `seq` function to generate the years.

```
ACTpop <- data.frame(year=seq(1917, 1997, 10),
                     ACT=c(3,8,11,17,38,103,214,265,320))
attach(ACTpop)
```

```
plot(year, ACT, type="l")
detach(ACTpop)
```

Another nicety is that mathematical expressions can be used in labelling plots. In the next example for you to try, the y-axis label is a mathematical expression.

```
p <- (0:100)/100
plot(p, sqrt(p*(1-p)), ylab=expression(sqrt(p(1-p))),
                                           type="l")
```

The function `help(plotmath)` has further details.

Adding text to points and other niceties

We use the data set *primates* that comes with the DAAG package. In this case we want to set up a working directory, *primates*, for this exercise. Carry out the following steps.

1. Set up a working directory, *Primates*, for this exercise. Use *File > New > Folder* in Windows Explorer to create the directory either within *E:\My Documents* or on a USB drive. You can always create it in *E:\My Documents\primates* and copy it to a USB drive later so that you can take it with you.
2. The data set is available in *Primates.data.rda* from the Computing files page of the web site. Download it and save it into the folder just created.
3. In the R Console window use *File > Change dir...* and browse to the *Primates* folder that you created in step 1.
4. In Tinn-R use *File > New* to open a new script file.
5. Enter the following commands into the Tinn-R script file and send them to R to create a first rough version of the brain weight versus body weight:

```
load("Primates.dat.rda")
attach(primates)
plot(Bodywt, Brainwt, xlim=c(0, 300))
text(x=Bodywt, y=Brainwt, labels=row.names(primates), adj=0)
detach(primates)
```

Note the use of `row.names` to extract these from *primates*. Also, `xlim` is specified to allow room for labels and `adj=0` implies left-adjusted text

6. Next we specify better x-axis and y-axis labels and move the labelling to one side of the points by including appropriate horizontal and vertical offsets. The latter is done by getting the character width (`chw`) and height (`chh`). Also `pch=16` is used to make the points plot as a solid dot. Modify the commands to the following commands and then send them to R.

```
attach(primates)
plot(x=Bodywt, y=Brainwt, pch=16, xlab="Body weight (kg)",
     ylab="Brain weight (g)", xlim=c(0, 300),
     ylim=c(0, 1500))
chw <- par()$cxy[1]
chh <- par()$cxy[2]
text(x=Bodywt+chw, y=Brainwt+c(-0.1, 0, 0, 0.1, 0)*chh,
```

```
labels=row.names(primates), adj=0)
detach(primates)
```

Multiple panels

A data set that comes with the DAAG library is called *possum*. It is available in the file `Possum.dat.rda` from the Resources web site and includes the variables `totlngh`, `age`, `sex` and `Pop` that record the total length, age, sex and whether the possums were from Victoria or elsewhere for 94 possums. We wish to examine the relationship between total length and age for each combination of sex and population. This is achieved using the following commands, the first of which loads the data frame *possum*:

```
load("Possum.dat.rda")
table(possum$Pop, possum$sex)
library(lattice)
xyplot(totlngh ~ age | sex*Pop, data=possum)
```

The `table` function displays the frequencies for the combinations of the levels of the two factors `Pop` and `sex`. We have used the form `y ~ x` to specify the y and x variables — it is the model formula form. The variables `sex` and `Pop`, after the “|”, are referred to as the conditioning variables: a plot is given for each of their levels combinations.

A.2. Rocket data

1. In this case we want to set up a working directory, *Rocket*, for this exercise. Use *File > New > Folder* in Windows Explorer to create the directory either within *E:\My Documents* or on a USB drive. You can always create it in *E:\My Documents\Rocket* and copy it to a USB drive later so that you can take it with you.
2. The commands for this exercise, given below, have been saved into *Rocket.r* and the associated data frame has been saved in *Rocket.dat.rda*. They are available from the Computing files page of the web site. Download them and save them into the folder created above.
3. Start Tinn-R and R. Use *File > Open* in Tinn-R to open the script file and then, in Tinn-R, set the current working directory to *Rocket* using the Tinn-R button for this.
4. Select the first three lines of the commands and use the *Send selection* button to send them to R. Got to the *R Graphics Device* window and copy Use *File > Copy to the clipboard > as a Metafile*. Paste it into a Word document.

```
load("Rocket.dat.rda")
attach(Rocket.dat)
plot(Thrust, Temp)
summary(Rocket.dat)
MnThrust <- mean(Thrust)
MnThrust
```

5. Select the last 3 lines and *Send selection* button to send them to R. Examine the summary statistics and mean in the R Console window.
6. Now close all R and Tinn-R. Do not save the workspace image.

A.3. Regression analysis of house price data

1. Create a new folder called *E:\My Documents\House Prices* or similar, depending on where you are creating folders.
2. Download *House.Prices.dat.rda* from the Computing files page of the web site and save it in the folder just created.
3. In the R Console window use *File > Change dir...* and browse to the House Prices folder that you created in step 1.
4. In Tinn-R use *File > New* to open a new script file.
5. Enter the following commands into the script file and send them to R.

```
load(House.Prices.dat)
attach(House.Prices.dat)
```

6. Use the *List all objects* button in Tinn-R to check that the data frame has been loaded. Also select *House.Prices.dat* somewhere in the Tinn-R window and click on the *List structure* button (or press the F2 key).
7. Now do an initial a scatterplot matrix by entering the following command into the script file and sending it to R.

```
pairs(House.Prices.dat, pch=16)
```

8. Use *File > Save as > Jpeg > 100% quality...* to save the scatterplot matrix in the current working directory as say *ScatterPlot.jpg*. If you do not save the plot at this point, it will be overwritten by the next plot.
9. Enter the following command into the script file and send them to R to produce the analysis and the residuals.

```
House.lm <- lm(Price ~ Age + Years, House.Prices.dat)
summary(House.lm)
residuals <- residuals(House.lm)
fitted <- fitted(House.lm)
plot(fitted, residuals)
```

10. Use *File > Save as > Jpeg > 100% quality...* to now save the residual versus-fitted values plot in the current working directory as say *ResvsFit.jpg*, again so that it is not overwritten by the next plot.
11. The following commands produce a quantile-quantile plot. Enter them into the script file and send them to R.

```
qqnorm(residuals)
qqline(residuals)
```

12. Use *> Save as > Jpeg > 100% quality...* to save the quantile-quantile plot in the current working directory as say *QQplot.jpg*. Also, click on the R Console window

and use *File > Save as* to save it in the current working directory as say *MultRegn.txt*.

13. Now to add the fitted values and residuals to the *House.Price.dat* data.frame. Enter the following commands:

```
detach(House.Prices.dat)
House.Prices.dat
  <- data.frame(House.Prices.dat, residuals, fitted)
```

Check that the two vectors have been added to *Houses.Prices.dat* by selecting the name anywhere in the script file in Tinn-R. Then click on the *List names* button in Tinn-R.

14. If the two vectors have been added to *Houses.Prices.dat*, delete them from the workspace using the following command

```
remove("residuals", "fitted")
```

15. Now close R, not saving the workspace image, and the files open in Tinn-R.

A.4. Entering factors into a Data window

Consider the following set of data that is a set of mileages and types of cars from a small study.

| Mileage | Type |
|---------|---------|
| 33 | Small |
| 33 | Small |
| 37 | Small |
| 20 | Sporty |
| 27 | Sporty |
| 19 | Sporty |
| 27 | Compact |
| 23 | Compact |
| 26 | Compact |

Enter this data into the data frame *TestData.dat* in R, with *Mileage* set up as a numeric vector and *Type* as a factor. In entering *Type*, enter codes 1, 2 and 3 for the values and the three names as labels.

Once you have the data entered, the following commands to produce the mean of each *Type*:

```
attach(TestData.dat)
tapply(Mileage, list(Type), mean)
```

You can also arrange to store the means along with the *Type* levels using the *aggregate* function as follows:

```
Mileage.Mean <- aggregate(Mileage, list(Type), mean)
```

A.5. Analysis of rat experiment

Consider an experiment in which the effects of three diets on rats are to be investigated. Suppose I have 6 rats to be fed one of three diets, 3 rats to be fed diet A, 2 diet B and 1 diet C. Then, at the end of the experiment, the experimenter measured the liver weight as a percentage of total body weight. The results of the experiment, in standard order, are as follows:

| Rat | 5 | 6 | 1 | 3 | 2 | 4 |
|-----------|-----|-----|-----|-----|-----|-----|
| Diet | A | A | A | B | B | C |
| Liver wt. | 3.3 | 3.1 | 2.9 | 3.2 | 3.4 | 2.7 |

1. Create a new working folder called *CRDRat*.
2. Create a new script file in Tinn-R.
3. Enter the Rat, Diet and Liver weights into a data frame and save them into the file *CRDRat.dat.rda* in the working directory using the `save` function.
4. The following commands make the data in *CRDRat.dat* available with having to include the name of the data frame in references to columns in the data frame and do an initial boxplot. Enter them into the script file and send them to R. You should see a boxplot in a Graph Device window.

```
attach(CRDRat.dat)
boxplot(split(LiverWt, Diet))
```

5. Make sure that the Graph sheet window is active and use *File > Save* to save the boxplot in *E:\CRDRat* as say *Boxplot.sgr*. If you do not save the plot at this point, it will be overwritten by the next plot.
6. Enter the following commands into the script window to produce the analyses by selecting just the commands that you want to run and sending them to R.

```
# AOV without Error
Rat.NoError.aov <- aov(LiverWt ~ Diet, CRDRat.dat)
summary(Rat.NoError.aov)
model.tables(Rat.NoError.aov, type = "means", se = T)
#
# AOV with Error
Rat.aov <- aov(LiverWt ~ Diet + Error(Rat), CRDRat.dat)
summary(Rat.aov)
```

7. Save the script file using *File > Save* in the working directory as say *CRDRat.r*.
8. Close R and save the workspace image. It will be saved as *.RData* in the current working directory.
9. Restart R and set the working directory to *Rocket*. Use *File > Load History..* to load the *.RData* file in the working directory.
10. Use the *List all objects* button in Tinn-R to check what objects have been reloaded.