# R and S-PLUS:
# BASIC INSTRUCTIONS

## Contents

## 1 GENERAL NOTES:

The instructions below work for both S-Plus and R. S-Plus is a very powerful statistics language from ATT Bell Laboratories. R is an open source copy of S-Plus and is available for Win9x and Unix/Linux operating systems. Whilst S-Plus (and thus R) is very flexible, expandable and virtually without limitations, because it is a programming language (albeit high-level), it does require basic programming concepts and has a steep learning curve. Having said this, the rewards for mastering the basic concepts of R are great. Other statistical packages such, such as SYSTAT and SPSS are inflexible, riddled with limitations and very expensive (not available to students off campus). R can be downloaded from:
`ftp://mirror.aarnet.edu.au/pub/CRAN /index.html`
**Warning**, R (and S-Plus) is a completely command and script based package. It is not possible to incorporate the huge array of functions etc into a usable menu-driven interface.
As R is a copy of S-Plus, reading documentation on either will provide information on usage. In particular Everitt (1994) and Venables and Ripley (1994) are excellent references. In addition, there is plenty of information available on the web and in help files that come with the distributions.

These instructions only cover some of the many options available for analyses and graphs. Once you are familiar with a particular type of analysis or graph, you might want to investigate other options. **Help** is always available. For each procedure described, an example of the commands needed are provided and additional scripts are placed in the Appendix. The commands take on the following structure:

- > this is the command prompt, you don't type this, it is already there.
- # this is a comments marker. It allows for comments in the script and every thing following it for that line will be ignored by R.
- `<-` is the equivalent of a '=' (equals sign).
- **Bold type** indicates that you should alter the text in here to correspond to the variable etc. that you are interested in.
- all white space is ignored. Occasionally, commands are split across lines in this document. When using the commands, remove the carriage returns.

R itself is driven by typed commands. Such an interface often frightens new users away, however, there are several good reasons for performing statistical analyses via commands

1. Forces the user to know more about what they are asking the software to perform. This minimizes the effects of the point-and-click (black box) syndrome where users click a couple of windows items, hit return, hope for the best and accepting any non error as indicating a correct outcome. Consequently, command driven statistics are potentially excellent teaching resources.

2. Enable easy storing of sequences of commands thereby permitting analyses to be replicated exactly (no matter how complex) at later date

3. Much quicker for developers to write and maintain code and therefore commands usually provide more flexible procedures than more graphical user interfaces

**Note:** these notes do not cover many of the more advanced techniques and issues of analysis. As the title implies, they are very much 'basic instructions'. More extensive examples and syntax is found within the Eworksheets.

## 2 INSTALLATION UNDER WINDOWS

The latest stable version of R can be downloaded from `http://users.monash.edu.au/~murray/stats/`. Installation is straight forward. Click on the **R installation** link and follow the prompts to download the *.exe file into a local directory on you computer (running Windows). Locate and double click on this file - this will run the setup software and install R on your computer (Note, you will be asked to confirm where R should be located, the default position is best!).
You should also download and install a number of packages that extend the functionality of R and provide a range of procedures that are useful for biologists (see section 4).

## 3 BRIEF INTRODUCTION

Although R for windows is a powerful statistical package, it is actual a high-level interpreted programming language (called R) which is modeled on S. Both R and S are object orientated languages and therefore everything in R is an object of some kind The basic data storage unit in R is called a *vector*. A vector is just an array of one or more entries (numbers, characters, etc). Hence a vector is an object. There are a number of types (or *classes*) of vector in R reflecting the nature of the data in the vector. There is a function called `c`, which is short (very short) for concatenate. This function generates a vector (collection of similar entries) of entries. For example, the following syntax generates a vector with the name `newvector` that contains two real (double precision or floats) numbers (8.4 and 2.1):

```
>a <- c(8.4, 2.1)
```

The statement (everything to the right of the R prompt, >) is evaluated when the ENTER key is pressed.
**A few necessary definitions**

| | |
|---|---|
| **Array** | A collection of one or more values of the same type (e.g. all numbers or all characters etc) arranged in one or more dimensions |
| **Vector** | A collection of one or more values of the **same type** (e.g. all numbers or all characters etc). `>c(1,2,3,4)` The above command generates a `numeric` *class* (type) of vector comprising of the values 1, 2, 3 and 4. Table 1 lists other common vector classes. |
| **Object** | Everything is an object. There are a number of types (or *classes*) of vector in R reflecting the nature of the data in |
| **Function** | A set of instructions carried out on one or more objects. Functions are typically used to perform specific and common tasks that would otherwise require many instructions. For example, the function `mean()` is used to calculate the arithmetic mean of the values in a given object (e.g. numeric vector) |

**Argument** Information parsed to a function to determine how the function should perform its task. Arguments are given between the brackets that proceed the name of the function. For example, the `mean` function requires at least one argument - the name of an object that contains the data from which the mean is to be generated

**Operator** Is a symbol that has a pre-defined meaning. Familiar operators include `+ - * / = < > <=` and `>=`, while less familiar operators include `==` (does the item on the left hand side of the `==` equal the item/value on the right hand side), `!=` (is the left hand side not equal to the right), `&&` (and) and `||` (or)

**Expression** A statement that is evaluated and used to produce a value that is then printed to the output and discarded

**Assignment** Evaluates an expression and assigns the result (value) to an object. The assignment operator `<-` is interpreted by R as 'evaluate the expression on the right hand side and assign it to the object on the left hand side. If the object on the left hand side does not already exist, then it is created, otherwise the objects contents are replaced. For example

```
>a <- 1
```

assigns the value 1 to an object called 'a'

**Formula**

Table 1 Object vector classes in R

| Vector class | Contains<br>`Example` |
|---|---|
| `integer` | whole numbers<br>`2:4` |
| `numeric` | real numbers<br>`c(8.4,2.1)` |
| `character` | letters<br>`c('A','Fish')` |
| `logical` | TRUE or FALSE<br>`c('FALSE','TRUE')` |
| `list` | list of objects<br>`list('A'=c(1,2),'B'=c('a', 'big'))`<br>This generates a list vector class that itself contains a numeric vector called A and a character vector called `B` |

## 3.1 Getting started

```
>help(mean)
OR
>?mean
```

Two alternatives for getting help on the function `mean()`

```
>q()
```

Quits R gracefully.

## 3.2 Command history

Using the up and down arrow keys on the keyboard to scroll backwards and fawards respectively through the previously enetered commands (command history), enables previous commands to be re-executed or modified and executed.

## 3.3 Variables - vectors

In biology, a variable is a collection of observations of the same type. For example, a variable might consist of the observed weights of individuals within a sample of 10 bush rats. Each item (or element) in the variable is of the same type (a weight) and will have been measured comparably (same techniques and units). Biological variables are therefore best represented in R by vectors.

Type out the following command and press enter

```
>4+2
```

The command is evaluatuated and the result (6) is printed on the default (screen) output device.

```
>4+2
[1] 6
```

The result is proceeded by **[1]** which indicates that this is the first element in the requested result. For such a simple example, the element lable is not necessary, however, its importance will become apparent later when the result includes a list of items (numbers or words).

Enter and explore the following commands (a brief description follows each entry):

```
>1:8
```

The : operator is interpreted as *'generate a sequence from the item on the left hand side to the item on the right hand side'*

```
>c(2.4, 3.5, 5.6, 6.2, 7.5, 8.4, 10.0, 12.1,
13.6, 13.9)
```

The c function concatenates a sequence of numbers into a vector. Hence the command above generates a number vector (containing 10 real numbers)

```
>x <- 1:10
```

Assign the sequence of numbers to a new object (an integer vector) called x. Note that when an object is assigned, its class is automatically determined by the contents assigned to it.

```
>x
```

Print to contents of the x object to the default device.

```
1. >wt <- c(126, 130, 121, 136, 139, 106, 111,
      102, 99, 115)
2. >wt
3. >class(wt)
```

1. Use the c function to generate a number vector (containing 10 real numbers) and assigns the vector to a new object (a number vector) called `wt`. In this case `wt` might represent the weights of 10 bush rats, and thus might represent a continuous variable.
2. Print to contents of the `wt` object
3. Use the `class()` function to determine (and print) which class of object the object `wt` belongs to

```
1. >X <- c(1, 2, 3, 2.4, 'a', 'high')
2. >X
3. >class(y)
```

1. Use the c function to a vector (specified with a collection of integers, real numbers and characters) and assigns the vector to a new object called X. Since all the items in a vector must be of the same type, all the the items are converted into characters and therefore the vector is a character vector. Note also that R is case sensitive and therefore x and X are not the same.
2. Print to contents of the X object
3. Use the `class()` function to determine (and print) which class of object the object X belongs to

### 3.3.1 Factorial variables

A specific type of character vector that is of particular importance in the analysis of grouped data is the `factor` class.

```
1. >SEX <- c('Male', 'Male', 'Male', 'Male',
     'Male', 'Female', 'Female', 'Female',
     'Female', 'Female')
2. >SEX
3. >SEX1 <- factor(SEX1)
4. >list(Sex=SEX, Sex1=SEX1)
5. >SEX <- factor(SEX, levels=c('Male',
     'Female'))
6. >SEX
7. >SEX <- gl(2,5,10,lab=c('Male','Female'))
8. >SEX
```

1. Use the c function to generate a character vector containing 6 items and assign it the name **SEX**
2. Print out the contents of the **SEX** character vector object.
3. Use the `factor` function to convert the character vector object into a factor object in which the unique

levels of the factor are defined. For comparison sake the result has been assigned to another factor object (**SEX1**), however it is more usual to reassign the result to the same object

4. Use the `list` function to print the contents of the original **SEX** character object and the **SEX1** factor object. The arguments `Sex=SEX` and `Sex1=SEX1` define names (can be any legal names you like) for the **SEX** and **SEX1** objects within the list and are used to identify them in the printed output.
   Note that the **SEX1** object also lists the levels within the factor in alphabetical order. Most statistical packages do this, however, it is usually not desirable as the alphabetical ordering of treatments is biologically meaningless.
5. Use the `levels=` argument to specifically assign the order of levels when assigning a factor object
6. Reprint the contents of the **SEX1** object. Notice that the order of the levels is more meaningful. The significance of the ordering will become more apparent later.
7. The `gl` function generates a factor vector and automatically sets the order of the levels according to the order defined by the argument (lab=c('Male','Female')) all in one relatively short command. The first argument of the `gl` function determines how many levels are in the resulting factor variable, the second determines how many times the items of each level should appear in succession and the third argument determines how many items in total should be in the vector. For simplicity as well as flexibility, the `gl` function is the author's preferred method for defining factors
8. Print the contents of the **SEX** factor vector

Note that the names given to vectors (variables) can comprise of virtually any sequence of letters and numbers provided the following rules are adhered to

1. Names must begin with a letter (not a number or an operator)
2. Names cannot include either _ (underscore) or a space character

Following is a list of naming recommendations

1. Names should reflect the content of the object. For example the name `wt` was used to represent a variable or weights
2. Although there is no restriction on name lengths, short names are quicker to type and therefore preferable
3. Separate words in names by a '.' (decimal point). For example the name `head.length` might be used to represent a variable of rat head lengths
4. Avoid names that are names of common functions (such as `mean`, `c` etc), as these can provide a source of confusion.

## 3.4 Data frames

A single biological variable is rarely collected and analyzed in isolation. Rather data are usually collected in sets of variables reflecting tests of relationships, differences between groups or as multiple characterizations. Consequently, data sets are best organized into collections of variables (vectors). Such collections are called *data frames* in R.
Data frames are generated by combining multiple vectors together whereby each vector becomes a separate column in the data frame. In for a data frame to represent the data properly, the sequence in which observations appear in the vectors (variables) must be the same for each vector and each vector should have the same number of observations. For example, the first observations from each of the vectors to be included in the data frame must represent observations collected from the same sampling unit.

```
1. >rats <- data.frame(sex=SEX, wt)
2. >rats
3. >row.names(rats) <- c('a', 'b', 'c', 'd',
      'e', 'f', 'g', 'h', 'i', 'j')
4. >rats
```

1. Use the `data.frame` function to generate a *data frame* object (named **rats**) from two vectors (**SEX** and **wt**). The argument **sex=SEX** is used to rename the vector **SEX** to **sex** within the data frame. Note that the first observations from the two vectors **wt** and **SEX** must represent observations collected from the same sampling unit (individual rat)
2. View the **rats** data frame (data set)

3. Use the `row.names` function to add labels (names) to each of the rows. These names are not necessary and purely act as a means to identify individual replicates
4. View the **rats** data frame (data set). Note the row names

Once a data frame has been formed, vectors (variables) are referred by the following syntax:

```
>frame$vector
```

where **frame** is the name of a data frame and **vector** is the name of a vector within that data frame
For example,

```
>rats$wt
```

Prints the contents of the vector **wt** within the **rats** data frame. Note also that the following two commands refer to two completely different variables;

```
>wt
>rats$wt
```

and that altering (or deleting) the vector **wt** will not alter the **rats$wt** vector and vica versa.

# 4 PACKAGES

R is a highly modular piece of software, consisting of a very large number of *packages*. Each package defines a set of functions that can be used to perform specific tasks. Packages also include of help files and example data sets and command scripts to provide information about the full use of the functions.

The modularized nature of R means that only the packages that are necessary to perform the current tasks need to be loaded into memory. This results in a very 'light-weight', fast statistical software package. Furthermore, the functionality of R can be easily extended by the creation of additional packages, rather than re-write all the software. As a result of this, and the open source license, new statistics are added to R nearly as soon as staticians publish them. New and revised packages can be freely downloaded from the CRAN website at any time.

## *4.1 Installing new packages*

To first install or update a package

### 4.1.1 Menus - windows

1. Download the package source (a zip file)
2. Perform the following menu sequence from *Rgui*

➠**Packages**

➠**Install package(s) from local zip files..**

Locate and select the name of the package and click the **OK**

### 4.1.2 Commands - UNIX

1. Download the package source (a tar.gz file)
2. From the directory containing the tar.gz file (that is from a terminal not currently engaged with R), type

```
R CMD INSTALL package.tar.gz
```

where **package.tar.gz** is the name of the package to be installed

## *4.2 Loading packages*

During the installation process of R, a large number of commonly used packages installed. When R is first started, only the `base` package is loaded. This package contains an extensive collection of functions for performing most of the basic data manipulation and statistical procedures.
To load additional packages for use during a session:

### 4.2.1 Commands

```
>library(package)
```

where **package** is the name of a pre-installed package to be loaded into memory.
For example:

```
>library(ctest)
```

loads a package called **ctest** which is a collection of extremely useful functions put together by Kurt Hornik for performing classical tests including t-tests, chi-squared tests and MannWhitney Wilcoxon tests.

#### 4.2.2 Menus

Perform the following menu sequence from *Rgui*
➠**Packages**

➠**Load package..**

Select the package from the list and click the OK

## 5 DATA FILES

Table 2 lists a fictitious data set that will be used to illustrate many basic statistical procedures in R

| ID | SEX | Weight | Head length |
|---|---|---|---|
| A | Male | 126 | 19 |
| B | Male | 130 | 21 |
| C | Male | 121 | 20 |
| D | Male | 136 | 23 |
| E | Male | 139 | 22 |
| F | Female | 106 | 16 |
| G | Female | 111 | 17 |
| H | Female | 102 | 15 |
| I | Female | 99 | 13 |
| J | Female | 115 | 16 |

### *5.1 Generating a new data set*

#### 5.1.1 Commands

The following lines of syntax generate 3 new variables.

```
1. >wt <- c(126, 130, 121, 136, 139, 106, 111,
      102, 99, 115)
2. >head.length <- c(19, 21, 20, 23, 22, 16,
      17, 15, 13, 16)
3. >sex <- gl(2,5,10,c('Male', 'Female'))
4. >rats <- data.frame(sex, wt, head.length)
5. >row.names(rats) <- LETTERS[1:10]
6. >rats
```

1. Use the c (concatenation) function to assemble a collection of real numbers into a continuous variable with 10 observations. Call this vector (a series of entities - like an array) **wt**
2. Assemble another continuous variable with 10 observations and call it **head.length**
3. Use the `gl` function to assemble a collection of category labels for a categorical variable and assign it the name **sex**.
4. Use the `data.frame` function to generate a data frame comprising of three vectors (based on **sex**, **wt** and **head.length**) and assign it to an object called **rats**
5. Use the `row.names` function to add row names to the data set. LETTERS is one of a handful of useful built in vectors defined in the *base* package and contains a sequence of capital letters from 'A' to 'Z'. Note that this step is not necessary, it purely assists in identifying individual sampling units (the individual rats in this case).
6. Print (view) the **rats** data frame

NOTE: the order in which values are placed into the variables is very important, as variables are usually collections of observations that are collected from the same sampling units.

There is also a primitive spreadsheet that can be used to enter and edit data from existing data frames.

```
>fix(frame)
```

where **frame** is the name of an exiting data frame. To use this spreadsheet to generate a new data frame, it is first necessary to define an empty data frame before calling `fix`. For example to generate the rats data set via the spreadsheet:

```
1. >rats <- data.frame(row.names=0, sex=0,
      wt=0, head.length=0)
      OR
2. >rats <- data.frame(row.names=
      LETTERS[1:10], sex=gl(2,5,10, c('Male',
      'Female')), wt=0, head.length=0)
3. >fix(rats)
```

1. Use the `data.frame` function to generate a new data frame (**rats**). Arguments such as **sex**=`0` generate blank vectors within the data set. If row names are to be entered using the spreadsheet, the `row.names` argument needs to appear as the first argument in the `data.frame` function.
2. The `data.frame` function can also include more complete vector definitions. In this case the row names and **sex** vector are predefined and **wt** and

head.length remain empty. Factor variables are often more efficiently and easily defined by commands whereas numberical vectors are often easier to fill using the spreadsheet.

3. Use the `fix` function to fill in the data frame

## 5.2 Opening an existing data set/file

R will accept (open and import) many different file types including SPSS, MINITAB and plain text.

### 5.2.1 PLAIN TEXT - comma delimited

```
1. >data <- read.table("filename", header=T,
      sep=",")
2. >data
```

1. The `read.table` function is used to load the file called **filename** (including full path if not in the current working directory) into a data frame format and is assigned the name **data**. The parameter `header=T` is used to retain column titles that are in the first row of the text file (if they are present) and the parameter `sep=","` specifies that the data in the text file are comma delimited. For space or tab delimited files use `sep=" "`.
2. List the contents of the resulting data frame. Note that the data is already in data frame format.

### 5.2.2 SPSS

```
1. >library(foreign)
2. >data <- read.spss("filename.sav")
3. >data
4. >data <- data.frame(data)
5. >data
```

1. The `library` function loads the `foreign` package which includes import filter functions for a number of data set formats including SPSS. If you have already loaded this library during the current R session (ie since you started the R program running), then there is no need to load it again, and this line can be ignored.
2. An SPSS data file with the name **filename.sav** (including full path if not in the current working directory) is loaded using the `read.spss` function into the native R format and is assigned the name **data**. Note that SPSS uses special type specifiers in variable names (such as a $ to indicate a categorical variable) that are not legal within the names of objects in R. R converts these characters into '.' (decimal points). While this is not a problem, it can be annoying and you might want to change the name of any vectors that are affected by this.
3. Lists the contents of the imported SPSS data set
4. Usually it is preferable to quickly convert the imported data into what is called a data frame since most people find this a more intuitive way of visualizing data.
5. List the contents of the resulting data frame.

## 5.3 Importing a data set/file from the clipboard

```
1. >data <- read.table("clipboard", header=T,
      sep="\t")
2. >data
```

1. The `read.table` function is used to load the clipboard data into a data frame format and is assigned the name **data**. The parameter `header=T` is used to retain column titles that are in the first row of the text file (if they are present) and the parameter `sep="\t"` specifies that the data in the text file are tab delimited (all data in the clipboard are tab delimeted).
2. List the contents of the resulting data frame. Note that the data is already in data frame format.

## 5.4 Coding factorial variables in imported data sets

Note, it is recommended that whenever a data set is imported from either a text file or other statistical package, categorical (factor) variables should be defined in R as factors (see section 3.3.1. This ensures that these variables are treated as categorical (rather than continuous) variables during subsequent analysis and is particularly important for variables whose levels names are numbers. It is also worth defining the order of the levels within the factor as well

```
>FACTOR <- factor(FACTOR,
levels=c('b','c','a'))
```

Where **FACTOR** is the name of a categorical variable and the **'b','c','a'** is an example of how the ordering of 3 levels within the factor can be defined To list or refer to a single variable (e.g. **DV** in a data frame (e.g. **dat** use the following syntax:

```
>dat$DV
```

To list the properties (attributes) of a variable or data frame use the following syntax:

```
>dat$DV
```

## 5.5 Saving a data file

### 5.5.1 Exporting as comma delimited text file

To save data (an array or list) in R, use the following commands:

```
>write.table(data,"filename",
     header=TRUE, sep=",")
```

Where **data** is the name of the data frame to export and **filename** is the name (including path) of file to be saved.. `Header=TRUE` retains variable names in the saved file and `sep=","` defines the data delimiter as a comma.

### 5.5.2 Storing data in R native format

To save data (an array or list) in R, use the following commands:

```
>dump("data","filename")
```

Where **data** is the name of the data frame (or other object) to save and **filename** is the name (including path) of file to be saved.

## 5.6 Transforming variables

Biological data often requires transforming or scaling.
Table 3 Common data transformations

| Transformation | Expression |
|---|---|
| $\log_e$ | `log(VAR)` |
| $\log_{10}$ | `log(VAR,10)` |
| $\log_{10}$ | `log10(VAR)` |
| $\sqrt{}$ | `sqrt(VAR)` |
| $\arcsin$ | `asin(sqrt(VAR))` |
| scale (mean=0,unit variance) | `scale(VAR)` |

where **VAR** is the name of the vector (variable) whose values are to be transformed.

```
1. >rats$logwt <- log(rats$wt,10)
2. >rats
```

1. Generate a new variable **logDV** that contains the $\log_{10}$ transformations of the variable **DV**
2. Print the **rats** data frame

## 5.7 Selecting subsets or subgroups of a data

When listing or referring to a variable for analysis, it is possible select a subset of the data by the following format:

### 5.7.1 Subsets within single vectors (variables)

Table 4 Listing or referencing subsets of the data

| Selection | Command |
|---|---|
| Values of **a** less than 50 | `a[a<50]` |
| The first 10 values in **a** | `a[1:10]` |
| The 20th to the 50th value of **a** | `a[20:50]` |

```
1. >var <- sample(20:100,10, replace=TRUE)
2. >var
3. >var[1:5]
4. >var[var>50]
```

1. Use the `sample` function to generate a *numeric* vector containing 10 random numbers (sampled with replacement) between 20 and 100

2. Print the **var** vector
3. Print elements 1 through to 5 of the **var** vector
4. Print only those values of **var** that are greater than 50

### 5.7.2 Subsets by factor levels

Returning to the rats data set

```
1. >male.rats.wt <- rats$wt[which(rats$sex ==
      'Male')]
2. >male.rats.wt <- rats$wt[rats$sex ==
      'Male']
3. >male.rats <- subset(rats,sex == 'Male')
4. >male.rats <- subset(rats$wt,rats$sex ==
      'Male')
5. >male.rats <- subset(rats,sex=='Male',
      select=wt)
```

1. Use the `which` function to generate a vector the values of **wt** in the data set **rat** for which the categorical variable **sex** is equal to `Male`. Note the use of ==. In this case it means "does the bit on the left side equal the value on the right side" (see the list of R operatators on page 4)
2. Same as above only shorter
3. Use the `subset` function to generate a new data frame (**male.rats**) from the original data frame (**rats**) containing the values of all the vectors (variables) that were present in the data set **rat** for which the categorical variable **sex** is equal to `Male`.
4. Use the `subset` function to generate a new data frame (**male.rats**) from the original data frame (**rats**) containing only the values of **wt** in the data set **rat** for which the categorical variable **sex** is equal to `Male`.
5. Alternative for above

## 5.8 By groups

It is also possible to perform calculations on a variable separately based on groupings defined in a categorical data variable. To request calculations be performed by groups.

```
>splitDat <- split(dataFrame,GROUPING)
```

Where **dataFrame** is the name of the data frame (data set), **GROUPING** is the name of a categorical (factor) variable in the data set. This function splits the data set into a number of smaller data sets (corresponding to the number of levels/groups within the variable **GROUPING**). The new name (**splitDat**) is used to refer to the collection of new smaller datasets. Note that the original data set (**dataFrame**) remains unaltered. Each subset can then be referred to using the following syntax:

```
1. >split.rats <- split(rats,sex)
2. >split.rats
3. >split.rats$Male
4. >split.rats$Male$wt
```

1. Use the `split` function to split a data set **dat** according to the levels in a categorical variable (**TREAT**)
2. List the levels of the split data set
3. List the **Male** subgroup of the **split.rats** data frame
4. List the values of variable **wt** within the **Male** subgroup of the **split.rats** data frame

# 6 SUMMARY STATISTICS

## 6.1 Univariate

To calculate basic summary statistics on a whole variable, use the following syntax:

```
>FUNCTION(VARIABLE)
```

Where **FUNCTION** is a statistical function (see the table 5 bellow) and **VARIABLE** is the name of a continuous variable

Table 5 Common summary statistic functions

| Statistic | Function command |
|---|---|
| Mean | `mean` |
| Variation | `var` |
| Standard deviation | `sd` |
| Number of observations | `length` |
| Median | `median` |
| Quantiles | `quantile` |

```
1. >mean(rats$wt)
2. >rats.mean.wt <- mean(rats$wt,trim=.1)
3. >rats.mean.wty
```

1. Use the `mean` function to calculate the arithmetic mean of the entire **wt** variable within the **rats** data frame
2. Use the `mean` function to calculate the trimmed mean (10%) value for the entire **wt** variable within the **rats** data frame and assign the value to a vector called **rats.mean.wt**
3. Print the value of the **rats.mean.wt** vector (trimmed mean).

### 6.2 Bivariate

```
>tapply(VARIABLE,GROUPING, FUNCTION)
```

Where **VARIABLE** is the name of the variable that you want to calculate the statistic for, **GROUPING** is the name of a categorical (factor) variable and **FUNCTION** is the name of the statistic that you want to perform (see table 5 for a list of common summary statistics). The function `tapply` applies a function to a variable separately for each level within a categorical variable. The above table lists some common summary statistics and the corresponding functions.

```
1. >rats.mean.wt <- tapply(rats$wt,rats$sex,
     mean)
2. >rats.mean.wt
3. >rats.sd.wt <- tapply(rats$wt,rats$sex, sd)
4. >rats.n.wt <- tapply(rats$wt,rats$sex,
     length)
5. >rats.se.wt <- rats.sd.wt/sqrt(rats.n.wt)
6. >dat.stats.wt <-cbind(mean=rats.mean.wt,
     SE=rats.se.wt)
7. >dat.stats.wt
```

1. Use the `tapply` function to calculate the mean of **wt** for each level of **sex** within the **rats** data set and store the results as an array of numbers called **rats.mean.wt**
2. Print the **rats.mean.wt** array
3. Use the `tapply` function to calculate the standard deviation of **wt** for each level of **sex** within the **rats** data set and store the results as an array of numbers called **rats.sd.wt**
4. Use the `tapply` function to calculate the number of **wt** observations for each level of **sex** within the **rats** data set and store the results as an array of numbers called **rats.n.wt**
5. Calculate the standard error of the mean of **wt** for each level of **sex** using the formula ($SE = sd/\sqrt{n})$) and store the results as an array of numbers in a variable called **rats.se.wt**
6. Use the `cbind` function (which binds elements - in this case arrays of numbers - by columns) to combine the two 1-dimensional arrays into a singe 2-dimensional array called **rats.stats.wt**. Note that the column names in this 2-dimensional array are defined as parameters in the `cbind` function.
7. List the resulting 2-dimensional array

# 7 TWO SAMPLE TESTS

### 7.1 Independent t-test

The following syntax performs a pooled (equal) variance student t-test

```
>t.test(VARIABLE~GROUPING,var.equal=TRUE)
```

Where **VARIABLE** is the name of a dependent variable and **GROUPING** is the name of a categorical (factor) variable with two levels. The `var.equal` parameter is set to TRUE for an equal variance t-test or FALSE for a separate variance t-test.

```
1. >rats.t <- t.test(rats$wt~rats$sex,
      var.equal=TRUE)
2. >rats.t
```

1. Perform a pooled variance t-test to compare the mean **wt** variable for the two levels of the **sex** factor and store the result in an object called **rats.t**

2. List the contents of the above object - e.g. the results of the t-test

## 7.2 Mann-Whitney-Wilcoxon test

The following syntax performs a Mann-Whitney-Wilcoxon test

```
>wilcox.test(VARIABLE~GROUPING)
```

Where **VARIABLE** is a continuous dependent variable and **GROUPING** is the name of a categorical (factor) variable with two levels.

```
1. >rats.wilcox <- wilcox.test(rats$wt ~
      rats$sex)
2. >rats.wilcox
```

1. Perform a Mann-Whitney-Wilcoxon (non-parametric, rank based test) to compare the ranks of the **wt** variable for each of the two levels of the **sex** factor and assign the resulting output to an object called **rats.wilcox**

2. List the Mann-Whitney-Wilcoxon output

## 7.3 Paired t-test

Paired data is usually represented a little differently, reflecting the paired manner in which the observations in the groups were collected. The following fictitious data set consists paired observations of the number of sea cucumbers recorded in permanent quadrats before and after a cyclone. Pairing data in this manner takes into consideration that sea cucumbers are distributed very patchily and that the number of sea cucumbers is likely to vary considerably between individual quadrats.

Table 2 A fictitious data set used to illustrate paired t-tests in R

| **Quadrat** | **Before** | **After** |
|---|---|---|
| Q1 | 12 | 9 |
| Q2 | 18 | 11 |
| Q3 | 9 | 8 |
| Q4 | 14 | 8 |
| Q5 | 22 | 17 |
| Q6 | 7 | 6 |

Generate the above data set using similar procedures as for the rats data set (see section 3.3). Hint, use the vector names **before** and **after**, use pair labels to generate the row names and call the data frame **cucumber**.

```
>t.test(VARIABLE1, VARIABLE2)
```

Where **VARIABLE1** and **VARIABLE2** represent the paired variables

```
1. >cucumber.pt <- t.test(before, after,
      data=cucumber, paired=TRUE)
2. >cucumber.pt
3. >matplot(t(cucumber),type='l')
```

1. Use the `t.test` function to perform a paired t-test to test the null hypothesis that the mean difference between **before** and **after** pairs equals 0 and store the result in an object called **cucumber.pt**

2. List the contents of the above object - e.g. the results of the paired t-test

3. Use the `matplot` and `t` functions to generate a graph that represents the Before-After trend for each quadrat (as lines). The `t` function is used to transpose the data.frame such that the variables are in rows as required by the `matplot` function.

The `type='l'` argument specifies that the graph should consist of lines rather than points

Incidently, it is also possible to analyze this data set when the data frame is setup in the format for independent t-tests. While this is not recommended it does illustrate the flexibility of R. In order to perform the t-test, we need to convert the data into the format used for classical hypothesis tests. That is one vector that lists the levels of the treatment and another variable that lists the observations. While it would be relatively easy to generate this data frame from scratch, for the purpose of introducing another useful R function, the existing paired data frame will be converted from what is known as 'wide' format to the 'long' format. The paired t-test will then be performed on this converted data frame.

```
1. >cuc <- reshape(cucumber, varying
      = list(names(cucumber)),
      times=names(cucumber),
      v.names='cucumbers', direction='long',
      ids=row.names(cucumber)
2. >cuc$times <- factor(cuc$time,
      levels=c('before,'after'))
3. >cuc
4. >cuc.pt <- t.test(cucumbers ~ time,
      data=cucumber, paired=T)
5. >cuc.pt
```

1. Use the `reshape` function to convert the wide format data frame (**cucumber**) into long format data (**cuc**). The first argument is the name of the data frame to convert, the second argument (`varying=list(names(`**`cucumber`**`))`) lists which variables are to be transformed from multiple columns into a single column. The third argument (`times=names(`**`cucumber`**`)`) determines the names of the levels for a new categorical variable from the variable names in the original data frame. The forth argument (`v.names=`**`'cucumbers'`**) provides a name for the newly generated dependent variable. The fifth argument (`direction='long'`) indicates that the data frame should be converted to `long` format. Finally, the last argument (ids=row.names(**cucumber**), defines the labels to put in a variable that is used to identify individual sampling units.
2. Use the `factor` function to define the vector **time** as a factor variable and specifically force the order of the levels
3. Use the `t.test` function to perform a paired t-test on the data within the newly formed data frame **cuc**
4. Print the results of the paired t-test

## 8 CORRELATION & REGRESSION

### 8.1 *Correlation*

Revisiting the **rats** data set that was generated in section 3.3. This will be used to examine the strength of the association between the weights and head lengths of the rats.

```
>cor.test(VARIABLE1, VARIABLE2,
    method="pearson")
```

Where **VARIABLE1** and **VARIABLE2** are two continuous variables. The parameter `method="pearson"` specifies Pearson's product moment correlation coefficient. Alternatives include `method='spearman'` and `method="kendall"` for Spearman rank $\rho$ and Kendall rank $\tau$ respectively.

```
1. >rats.cor <- cor.test(wt,head.length, data
      = rats, method='pearson')
2. >rats.cor
```

1. Use the `cor.test` function to calculate the Pearson's product moment correlation coefficient and associated p-value for the association of **wt** and **head.length** in the data set **rats** and assign the output to an object named **rats.cor**. Note that as the direction of causality is not implied in correlation, it does not matter which of the vectors (variables) is listed in the `cor.test` function first
2. View the correlation output

### 8.2 *Regression*

To illustrate regression, the following fictitious data set will be used. Samples were collected from 12 lakes to examine the relationship between the number of gastropod (snail) species and the level of salinity. **Species** is

the dependent (response) variable and **salinity** is the independent (predictor) variable.
Table 7 Fictitious data set to illustrate linear regression in R

| Lake | species | salinity |
|---|---|---|
| 1 | 5 | 21.5 |
| 2 | 2 | 43.7 |
| 3 | 4 | 19.4 |
| 4 | 8 | 19.3 |
| 5 | 5 | 22.8 |
| 6 | 10 | 6.7 |
| 7 | 9 | 10.5 |
| 8 | 13 | 5.5 |
| 9 | 8 | 2.0 |
| 10 | 10 | 12.7 |
| 11 | 7 | 27.8 |
| 12 | 20 | 0.8 |

Generate the above data set using similar procedures as demonstrated in section 3.3. Hint, use the vector names **species** and **salinity**, use pair labels to generate the row names and call the data frame **snails**.

```
>lm(DEPENDENT~INDEPENDENT)
```

Where **DEPENDENT** is a continuous dependent variable and **INDEPENDENT** is a continuous independent variable (with respect to the dependent variable). The `lm` function evaluates any linear model. The statement of the form **DEPENDENT~INDEPENDENT** is a linear model.

```
1. >snails.lm <- lm(snails$species ~
      snails$salinity)
2. >snails.lm <- lm(species ~ salinity,
      data=snails)
3. >anova(snails.lm)
4. >summary(snails.lm)
5. >influence.measures(snails.lm)
6. >par(mfrow=c(2,2))
7. >plot(snails.lm)
8. >par(mfrow=c(1,1))
9. >plot(species ~ salinity, data=snails)
10. >abline(snails.lm)
```

1. Use the `lm` function to evaluate the linear relationship between **species** and **salinity** in the **snails** data frame and assign the output object to the name **snails.lm**. In the process of performing the linear regression, many results/diagnostics are calculated throughout the process - far too many to be displayed in full. These are all stored in the object pointed to be the variable **snails.lm** and a number of different functions are used to display or summarize the different linear model outputs.
2. An alternative to the previous command. Using the `data=` parameter prevents having to repeatedly prepend each variable name with the data frame name
3. The `anova` function will summarize the Analysis of Variance (partitioning of total variation in the dependent variable into components explained and unexplained by the linear model)
4. The `summary` function will summarize the estimated regression coefficients and associated hypothesis tests
5. The `influence.measures` function lists a range of regression diagnostics used to evaluate the influences of each observation on the linear regression outcome. The most important are `cook.d` (Cooks D) and `hat` (leverage)
6. Use the `par` function to setup the graphics device (Graph window) ready to accept 4 graphs on one page with a 2x2 layout
7. Produces 4 diagnostic plots (Plot 1 = residual plot, Plot 2 = normal Q-Q plot, Plot 3 = scale-local plot & Plot 4 = Cook's D plot)
8. Use the `par` function to restore the graphics device to display only single graphs per page
9. Use the `plot` function to plot to generate a scatterplot of **species** against **salinity**
10. Use the `abline` function to fit a line of best fit through the data using the regression coefficients to determine the slope and intercept of the line

## 9 ANOVA

To illustrate simple ANOVA, the following fictitious data set will be used. The diversity of arboreal mammals occupying forest patches of differing disturbance frequencies ('High', 'Low' and 'Natural') were examined across a large scale forest matrix. Disturbance frequencies were replicated 4 times and therefore a total of 12 different forest patches were examined.

Table 9 Fictitious data set to illustrate ANOVA in R

| High | Low | Natural |
|---|---|---|
| 1 | 3 | 7 |
| 3 | 4 | 6 |
| 1 | 3 | 6 |
| 0 | 2 | 5 |

Note the above data set has been presented in 'wide' format to conserve space. Generate the above data set using similar procedures as demonstrated in section 3.3. Hint, use the vector names **treat** and **abund** to represent the factorial variable (listing the levels of the disturbance frequency) and dependent variable (number of arboreal mammal species). Call the data frame **mammals** and provide unique patch names for the row names.

```
1. >abund <- c(1,3,1,0,3,4,3,2,7,6,6,5)
2. >treat <- gl(3,4,12,lab=c('High', 'Low',
     'Natural'))
3. >mammals <- data.frame(treat, abund,
     row.names=paste('Patch', 1:12, sep='-'))
4. >mammals
```

1. Use the c function to generate the dependent variable
2. Use the `gl` function to generate the factor variable
3. Use the `data.frame` function to generate the data frame. Note that the argument `row.names=paste('Patch', 1:12, sep='-')` is used to generate the row names during the data frame generation process. The `paste` function was used to generate the unique labels (`Patch-1` through to `Patch-12`)

## *9.1 Single factor ANOVA*

```
>aov(DEPENDENT~GROUPING)
```

Where **DEPENDENT** is a continuous dependent variable and **GROUPING** is an independent categorical variable with 2 or more levels. The `aov` function calls the `lm` function (see Regression) and therefore evaluates a linear model. The statement of the form **DEPENDENT~GROUPING** is a linear model.

```
1. >mammals.aov <- aov(abund ~ treat,
     data=mammals)
2. >anova(mammals.aov)
3. >influence.measures(mammals.aov)
4. >par(mfrow=c(2,2))
5. >plot(mammals.aov)
6. >par(mfrow=c(1,1))
```

1. Use the `aov` function to compare the group means for the variable **abund** in the **mammals** data frame and assign the output object to the name **mammals.aov**. In the process of performing the analysis of variance (as with other linear models), many results/diagnostics are calculated. These are all stored in the object pointed to be the variable **mammals.aov** and a number of different functions are used to display or summarize the different linear model outputs. ANOVA is a specific form of linear model that operates on categorical predictors. Likewise, the `aov` function is a specific type of `lm` function that partitions variance in to explained and unexplained and builds analysis of variance tables. While, the same result would be obtained using the `lm` function, the resulting model coefficients are of little value in ANOVA. Furthermore, the resulting `aov` object that is generated contains parameters lacking from the more general `lm` object that facilitate subsequent post-hoc and planned comparisons testing
2. The `anova` function will summarize the Analysis of Variance (partitioning of total variation in the dependent variable into components explained and unexplained by the linear model). Note that as `aov` is a specific form of `lm` in which the estimated coefficients are usually of little value, the functions `anova` and `summary` produce the same output when applied to an `aov` object
3. The `influence.measures` function lists a range of regression diagnostics used to evaluate the influences of each observation on the linear regression outcome. The most important are `cook.d` (Cooks D) and `hat` (leverage)
4. Setup the graphics window such that 4 graphs will be produced on the one page

5. Produces 4 diagnostic plots (Plot 1 = residual plot, Plot 2 = normal Q-Q plot, Plot 3 = scale-local plot & Plot 4 = Cook's D plot)

6. Restore the graphics window to the default setting of a single graph per page

## 9.2 Post Hoc Tukey's test

```
>summary(simtest(DEPENDENT ~ GROUPING,
data=data, type="Tukey"))
```

Where `simtest` is a function that calculates a range of post-hoc tests (including the Tukey Honest Significant Differences).

```
1. >library(multcomp)
2. >mammals.tuk <- summary(simtest(abund ~
     treat, data=mammals, type="Tukey")
3. >mammals.tuk
```

1. Use the `library` function to load the `multcomp` package (if not already loaded). Note that the `multcomp` package itself requires the `mvtnorm` package, so ensure that the latter is also installed (although not necessarily loaded as loading the `multcomp` package will automatically load it)

2. Use the `simtest` function to calculate the Tukey Honest Significant Differences arising from the model supplied (**abund ~ treat**)

3. List the differences and intervals

## 9.3 Planned comparisons

Planned comparisons are performed following a significant global ANOVA
This is a two stage process;

```
>contrasts(GROUPING) <- cbind(c(numeric
list),...)
```

where (**GROUPING**) is the name of the factorial variable and **numeric list** is a list of contrast coefficients for the factorial variable.

```
1. >contrasts(mammals$treat <-
     cbind(c(1,-1,0), c(1,1,-2))
2. >round(crossprod(contrasts(mammals$treat)),
     2)
```

1. Use the `cbind` function (which binds data together in columns) to generate a list of contrast coefficients. The number of entries must not exceed 1 minus the number of levels within the categorical variable.

2. Use the `crossprod` function to check that the defined contrasts are orthogonal.

Fit or re-fit and summarize the global ANOVA:

```
>AOV <- aov(DEPENDENT ~ GROUPING, data=data)
>summary(AOV, split = list(GROUPING =
     list("LABEL1"=1, "LABEL2"=2, ...)))
```

Where the `aov` model is run with the intention of performing a few specific planned comparisons. **DEPENDENT** is a dependent variable, **GROUPING** is a categorical (factor) variable

```
1. >mammals.aov <- aov(abund~treat,
     data=mammals)))
2. >summary(mammals.aov, split = list(treat
     = list("High vs Low"=1, "(High  Low) vs
     Natural"=2)))
```

1. Use the `aov` function to compare the group means for the variable **aov** in the **mammals** data frame and assign the output object to the name **mammals.aov**. The model incorporates the predefined contrasts to define which planned comparisons to perform. The resulting ANOVA `aov` object is assigned the name **mammals.aov**

2. Use the `summary` function to view the results of the ANOVA and planned comparisons (**mammals.aov**). The `split` argument is used to generate the labels that will be used in the table to denote the specific comparison tests. These entries should be in the same order as the defined contrasts. Each entry takes on the form such as **"LABEL1"**=c(**CONTRASTS1**) where **LABEL1** is a label (string) that will be used to denote the comparison in subsequent tests, and **CONTRASTS1** is

a set of contrast coefficients. The number of entries must not exceed 1 minus the number of levels within the categorical variable.

## 9.4 Factorial ANOVA

To illustrate factorial ANOVA, the following fictitious data set will be used. The water loss of leaves at 2 temperatures (20° and 30°) and 3 humidities (45, 75 and 100%)

Table 9 Fictitious data set to illustrate multifactor ANOVA in R

| **Temp** | **20** | | | **30** | | |
|---|---|---|---|---|---|---|
| **Humidity** | **45** | **75** | **100** | **45** | **75** | **100** |
| | 66 | 72 | 100 | 82 | 86 | 100 |
| | 54 | 82 | 99 | 82 | 90 | 94 |
| | 69 | 86 | 92 | 78 | 86 | 98 |
| | 61 | 86 | 100 | 80 | 88 | 99 |

Note the above data set has been presented in 'wide' format to conserve space. Generate the above data set using similar procedures as demonstrated in section 3.3. Hint, use the vector names **temp**, **humidity** and **water** to represent the 2 factorial variables (temperature and humidity) and dependent variable (water loss by the leaves). Call the data frame **leaves** and provide unique leaf names for the row names.

```
1. >water <- c(66, 54, 69, 61, 72, 82, 86, 86,
     100, 99, 92, 100, 82, 82, 78, 80, 86, 90,
     86, 88, 100, 94, 98, 99)
2. >temp <- gl(2,4,24,lab=c(20, 30))
3. >humidity <- gl(3,4,24,lab=c(45, 75, 100))
4. >leaves <- data.frame(temp, humidity,
     water, row.names=paste('Leaf', 1:24,
     sep=''))
5. >leaves
```

```
>aov(DEPENDENT~FACTOR1*FACTOR2)
```

Where the `aov` function is used to evaluate the effects of two factors (**FACTOR1** and **FACTOR2**) on the dependent variable **DEPENDENT**

```
1. >leaves.aov <- aov(water ~ temp * humidity,
     data=leaves)
2. >anova(leaves.aov)
3. >influence.measures(leaves.aov)
4. >par(mfrow=c(2,2))
5. >plot(leaves.aov)
6. >par(mfrow=c(1,1))
```

1. Use the `aov` function to compare the group means for the variable **water** in the **leaves** data frame and assign the output object to the name **leaves.aov**.

2. The `anova` function will summarize the Analysis of Variance (partitioning of total variation in the dependent variable into components explained and unexplained by the linear model). Note that as `aov` is a specific form of `lm` in which the estimated coefficients are usually of little value, the functions `anova` and `summary` produce the same output when applied to an `aov` object

3. The `influence.measures` function lists a range of regression diagnostics used to evaluate the influences of each observation on the linear regression outcome. The most important are `cook.d` (Cooks D) and `hat` (leverage)

4. Setup the graphics window such that 4 graphs will be produced on the one page

5. Produces 4 diagnostic plots (Plot 1 = residual plot, Plot 2 = normal Q-Q plot, Plot 3 = scale-local plot & Plot 4 = Cook's D plot)

6. Restore the graphics window to the default setting of a single graph per page

As for single factor ANOVA (see section 9.1), except enter 2 or more factors in **Factors** box. SYSTAT will then do a fully factorial ANOVA. To plot interactions, see section 12.4 below. Note that neither planned comparisons or post hoc tests can be performed on multifactor ANOVA's. If the interaction term is not significant, then break the analysis down and explore each of the simple main effects separately. If the interaction is significant, explore the effects of one factor separately for each level of the other factor(s).

### 9.4.1 Unbalanced multifactor designs

The balance of a design (model) is assessed with the following command;

```
>!is.list(replications(model formula,
data=data))
```

where **model formula** is the model formula you intend to use to fit the linear model. If the above command returns a 'FALSE', then the model is not balanced (unequal replication), otherwise it is balanced.

If the model is unbalanced, it is necessary to use Type II SS. This is done with following commands;

```
1. >library(biology)
2. >>AnovaM(AOV, type="II")
```

where **AOV** is the name of the fitted model.

```
1. >leaves.aov <- aov(water ~ temp * humidity,
     data=leaves)
2. >library(biology)
3. >AnovaM(leaves.aov, type="II")
```

1. Use the aov function to compare the group means for the variable **water** in the **leaves** data frame and assign the output object to the name **leaves.aov**.
2. Load the biology package
3. The AnovaM function will summarize the Analysis of Variance (partitioning of total variation in the dependent variable into components explained and unexplained by the linear model) using Type II sums of squares.

## 9.5 Nested ANOVA

To illustrate nested ANOVA, the following fictitious data set will be used. The data were collected to examine the effects of Phosphorus addition on the growth rate of *Banksia* seedlings. Seedlings were planted in 6 plots, 3 of which had Phosphorus added. The growth of three seedlings per plot were measured.

Table 11 Fictitious data set to illustrate nested ANOVA in R

| Treatment | Added | | | Control | | |
|---|---|---|---|---|---|---|
| Plot | 1 | 2 | 3 | 4 | 5 | 6 |
| | 12 | 9 | 17 | 5 | 4 | 7 |
| | 15 | 11 | 9 | 7 | 12 | 11 |
| | 13 | 13 | 10 | 8 | 8 | 10 |

Again to conserve space, the above data set has been presented in 'wide' format. Generate the above data set using similar procedures as demonstrated in section 3.3. Hint, use the vector names **treat**, **plot** and **growth** to represent the factor variables (Phosphorus added or control), the nesting factor (plots) and dependent variable (growth rate of seedlings). Call the data frame **seedlings** and provide unique seedling identities for the row names.

```
1. >growth <- c(12, 15, 13, 9, 11, 13, 17, 9,
     10, 5, 7, 8, 4, 12, 8, 7, 11, 10)
2. >treat <- gl(2,9,18,lab=c('Added',
     'Control'))
3. >plot <- gl(6,3,18,lab=c(1:6),
     row.names=paste('Seedling',1:6,sep='-'))
4. >seedlings <- data.frame(treat, plot,
     growth)
5. >seedlings
```

```
>aov(DEPENDENT ~ GROUPING + Error(GROUPING
     %in% NEST))
```

Where the aov function is used to evaluate the effects of a fixed factor **GROUPING** and a random nesting factor **NEST** on the mean **DEPENDENT** variable.

```
1. >seedlings.aov <- aov(growth ~ treat +
     Error(treat %in% plot), data=seedlings)
2. >summary(seedlings.aov)
3. >summary(aov(growth ~ treat/plot,
     data=seedlings))
4. >influence.measures(seedlings.aov$[2])
5. >plot(seedlings.aov[[2]]$residuals,
     seedlings.aov[[2]]$fitted)
```

1. Use the aov function to compare the group means for the variable **growth** in the **seedlings** data frame

using a nested ANOVA model and assign the output object to the name **seedlings.aov**. Note that the expression `+ Error(treat %in% plot)` defines the correct error term for the effects of the main fixed effect (**treat**) when the nesting factor (**plot**) is a random factor.

2. List the resulting ANOVA table. Note that the effect of the nesting factor is not calculated. However, as this is a random factor introduced purely for the purpose of reducing some of the unexplained variability, a formal analysis of its effect is of little value.

3. Should you have a desperate need to formally test the effect of the nested random factor, run the model as if the nesting factor was fixed - just remember to ignore the test of the main fixed effect as this will be incorrect.

4. The `influence.measures` function lists a range of regression diagnostics used to evaluate the influences of each observation on the linear regression outcome. Since the **seedling.aov** object has multiple error strata, it is necessary to define which strata the diagnostics should be generated for. The most important are `cook.d` (Cooks D) and `hat` (leverage)

5. Produces a diagnostic residual plot

## 9.6 Randomized Block

To illustrate Randomized block ANOVA, the following fictitious data set will be used. The contribution of stream macro-invertebrates to leaf litter decay was investigated using submerged packs of leaves that were either enclosed in a fine exclusion mesh (to prevent access by invertebrates) or not completely enclosed (Controls). As streams are very patchy environments, treatments were spatially blocked in an attempt to reduce some of the unexplained variation.
Table 11 Fictitious data set to illustrate nested ANOVA in R

| Block | Exclusion | Control |
|---|---|---|
| 1 | 43 | 74 |
| 1 | 28 | 56 |
| 1 | 35 | 58 |
| 1 | 51 | 88 |
| 1 | 37 | 63 |
| 1 | 39 | 78 |

Again to conserve space, the above data set has been presented in 'wide' format. Generate the above data set using similar procedures as demonstrated in section 3.3. Hint, use the vector names **treat**, **block** and **prop** to represent the factor variable (invertebrates Excluded or Control), the blocking factor (blocks) and dependent variable (proportion of leaf material decayed). Call the data frame **decay**.

```
1. >prop <- c(43, 28, 35, 51, 37, 39, 74, 56,
     58, 88, 63, 78)
2. >treat <- gl(2,6,12,lab=c('Exclusion',
     'Control'))
3. >block <- gl(6,1,12,lab=LETTERS[1:6])
4. >decay <- data.frame(treat, block, prop)
5. >decay
```

```
>aov(DEPENDENT~GROUPING + BLOCK)
```

Where the `aov` function is used to evaluate the effects of a fixed factor **GROUPING** and a blocking factor **BLOCK** on the mean **DEPENDENT** variable.

```
1. >decay.aov <- aov(prop ~ block + treat,
     data=decay)
2. >summary(decay.aov)
3. >influence.measures(decay.aov)
4. >par(mfrow=c(2,2))
5. >plot(decay.aov)
6. >par(mfrow=c(1,1))
```

1. Use the `aov` function to compare the group means for the variable **prop** in the **block** data frame using a randomized block ANOVA model and assign the output object to the name **decay.aov**. Although it doesn't strictly matter which factor (the treatment or the blocking) appears first in the model formula, putting the blocking factor first reminds you of the position of each of the factors in the design hierarchy

2. List the resulting ANOVA table.

3. The `influence.measures` function lists a range of regression diagnostics used to evaluate the influences of each observation on the linear regression

outcome. The most important are `cook.d` (Cooks D) and `hat` (leverage)

4. Setup the graphics window such that 4 graphs will be produced on the one page

5. Produces 4 diagnostic plots (Plot 1 = residual plot, Plot 2 = normal Q-Q plot, Plot 3 = scale-local plot & Plot 4 = Cook's D plot)

6. Restore the graphics window to the default setting of a single graph per page

### 9.6.1 Sphericity

For methods of dealing with sphericity, please refer to the Eworksheets.

## *9.7 Split-plot & Repeated Measures*

### 9.7.1 Factors fixed, plot random

```
>aov(DEPENDENT ~ BETWEEN + Error(PLOT %in%
BETWEEN) + WITHIN + BETWEEN:WITHIN)
```

Where **BETWEEN** is a fixed between plot (or subject) effect, **PLOT** is the random plot (subject) effect, **WITHIN** is a fixed within plot (subject) effect and **BETWEEN:WITHIN** is the within plot (subject) interaction between the two fixed factors. The symbol **:** denotes a cross (interaction term) and the symbol **%in%** denotes nesting. Note that **:** can also be used to denote nesting.

To illustrate Split-plot ANOVA in R, a modified version of the **decay** data set will be used. If surface shade cloth was fitted above half (randomly allocated) of the plots (to shade from the sun) and the other three plots were left exposed, then it becomes a split-plot design. The shading treatment becomes the between plots (blocks) factor and the exclusion treatment and exclusion treatment by shading treatment become the within plot factors. To conform with popular naming conventions, the name **block** will be changed to **plot**

```
1. >decay$plot <- decay$block
2. >shade <- gl(2,3,12, lab=c('Shaded',
     'Exposed'))
3. >decay <- data.frame(decay,shade)
4. >decay.aov <- aov(prop ~ shade + Error(plot
     %in% shade) + treat + shade:treat,
     data=decay)
     OR
5. >decay.aov <- aov(prop ~ shade*treat +
     Error(plot %in% shade), data=decay)
6. >>summary(decay.aov)
7. >summary(aov(prop ~ shade + plot %in% shade
     + treat + shade:treat, data=decay))
     OR
8. >1-pf(3.06/1.807,12,36)
9. >influence.measures(decay.aov[[2]])
10. >influence.measures(decay.aov[[3]])
11. >plot(decay.aov[[2]]$res ~
     decay.aov[[2]]$fit)
12. >plot(decay.aov[[3]]$res ~
     decay.aov[[3]]$fit)
13. >interaction.plot(decay$shade,
     decay$treat, decay$prop)
```

1. Copy the variable **block** to the variable **plot**. This is purely to conform to the different naming convention used for randomized block versus split plot designs, but of course has absolutely no impact on the analysis

2. Use the `gl` function to generate a categorical variable (assigned the name **shade**) with two levels (`'Shaded', 'Exposed'`) each item replicated once in succession to generate a maximum of 12 entries. This can be a fictitious, independent, fixed, between plot (subject) effect. Note again that the order of the observations is important.

3. Use the `c` function to generate a continuous variable (assigned the name **DV**) with 12 observations. This can be a fictitious dependent variable.

4. Use the `data.frame` function to assimilate the 4 variables into a single data set and assign it the name **plot**

5. Use the `aov` function to compare the group means for the variable **prop** in the **decay** data frame using

a split-plot ANOVA model where the between and within plot (subjects) effects (**shaded** and **treat** respectively) are fixed and the plot (**plot**) is a random factor. Assign the output object to the name **decay.aov**. The `aov` object created contains 3 strata that correspond to the 3 error strata for the model. Stratum one contains an estimate of the overall mean. However, as the degrees of freedom of this estimate is 0, this stratum is usually ignored. Stratum two lists the between subject (plot) effects and stratum 3 lists the within subject (plot) effects.

6. Alternative, shorthand syntax for performing the same split-plot ANOVA described above. The term **shaded*treat** will expand to **shaded + treat + shaded:treat**
7. List the resulting ANOVA table.
8. Should you have a desperate need to formally test the effect of the nested random factor, run the model as if the nesting (plot or subject) factor was fixed - just remember to ignore the test of the main between fixed effect as this will be incorrect.
9. Alternatively, to get a quick estimate of the effect of the nesting effect, use the `pf` function (probably distribution function for an F-ratio) to determine the probability of obtaining an F-ratio (**3.06/1.807**) or greater from a F distribution with **12** and **36** degrees of freedom if the null hypothesis is true and there is no effect of the nesting factor. The values **3.06** and **12** represent the estimate MS and degrees of freedom respectively for the effect of the nesting factor and the values **1.807** and **36** represent the residual MS and degrees of freedom respectively.
10. Use the `influence.measures` function to lists a range of regression diagnostics used to evaluate the influences of each observation on the estimates of between subjects (plots) effects (stratum 2). The most important are `cook.d` (Cooks D) and `hat` (leverage)
11. Use the `influence.measures` function to lists a range of regression diagnostics used to evaluate the influences of each observation on the estimates of within subjects (plots) effects (stratum 3). The most important are `cook.d` (Cooks D) and `hat` (leverage)
12. Create a residual plot for the between plots (subject) effect (stratum 2)
13. Create a residual plot for the within plots (subject) effect (stratum 3)
14. Use the `interaction.plot` to represent the within subject (plots) interaction

#### 9.7.2 Sphericity

For methods of dealing with sphericity, please refer to the Eworksheets.

# 10 Frequency analysis

## *10.1 Goodness of fit tests*

```
>chisq.test(COUNTS,p=c(PROPORTIONS))
```

Where the `chisq.test` function is used to compare the ratio in the **COUNTS** variable (observed ratio) to the proportions ratio specified in **PROPORTIONS** (Expected ratio)

```
>sex <- c(36,44)
>chisq.test(sex)
>chisq.test(sex,p=c(4/9,5/9))
```

1. Use the c function to generate an array of counts to represent the observed ratio and assign it the name **sex**. For example the observed sex ratio might be 36:44 (male:female)
2. Use the `chisq.test` function to evaluate whether the observed ratio (**sex**) differs significantly from a 1:1 ratio
3. Use the `chisq.test` function to evaluate whether the observed ratio (**sex**) differs significantly from a proportional ratio of **4/9,5/9** (in this case 4:5)

## *10.2 Contingency tables*

To illustrate contingency tables, the following fictitious data set will be used. 123 birds observed during a survey were cross-classified according to 2 variables, Height of canopy (ground level, middle canopy or treetops) and tree type (Native or Exotic)

Table 12 Fictitious data set to illustrate contingency tables in R

| Tree type | Ground | Middle | Tree top |
|---|---|---|---|
| **Native** | 12 | 24 | 19 |
| **Exotic** | 26 | 31 | 11 |

The data above are presented in collated table format. Generate the above data set using similar procedures as demonstrated in section 3.3. Hint, use the vector names **height** and **type** to represent the factor variables (Tree height and type), and the name **count** for dependent variable (number of birds in each cross-classification). Call the data frame **birds**.

```
1. >height <- gl(3,1,6, labels = c('Ground',
     'Middle', 'Top'))
2. >type <- gl(2,3,6, labels = c('Native',
     Exotic'))
3. >counts <- c(12,24,19,26,31,11)
4. >birds <- data.frame(height, type, counts)
```

```
chisq.test(XTABLE,correct=F
```

Where the function `chisq.test` is used to test the independence of 2 or more categorical variables defined in the contingency table (**XTABLE**

```
1. >birds.tab <- xtabs(counts ~ height + type,
     data=birds)
2. >birds.chi <- chisq.test(birds.tab,
     correct=F)
3. >birds.chi
4. >birds.chi$residuals
```

1. Use the `xtabs` function to convert the data in the data frame **birds** into a two-way contingency table assigned the name **birds.tab**
2. Use the `chisq.test` function to test for independence between the 2 categorical variables in the contingency table **birds.tab**
3. List the output of the Pearson's $\chi^2$ test
4. List the standardized residuals in table format

# 11 MULTIVARIATE ANALYSIS

To illustrate Multivariate analysis in R, the following fictitious data set will be used. The abundance of 5 species of cockroach were measured from 6 sites

Table 13 Fictitious data set to multivariate analysis in R

| Sites | Sp.a | Sp.b | Sp.c | Sp.d | Sp.e |
|---|---|---|---|---|---|
| BCI | 0 | 14 | 28 | 0 | 68 |
| LC | 0 | 38 | 4 | 0 | 29 |
| FOR | 0 | 1 | 1 | 0 | 0 |
| BOQ | 0 | 1 | 0 | 0 | 0 |
| MIR | 0 | 4 | 18 | 0 | 11 |
| CORG | 0 | 0 | 0 | 0 | 24 |

The data above are presented with variable (species) in columns and objects or sampling units (SU's) in rows. Generate the above data set using similar procedures as demonstrated in section 3.3.

```
1. >Sp.a <- c(0,0,0,0,0,1)
2. >Sp.b <- c(14,38,1,1,4,0)
3. >Sp.c <- c(28,4,1,0,1,0)
4. >Sp.d <- c(7,0,0,0,0,0)
5. >Sp.e <- c(68,29,0,0,11,24)
6. >cockroach <- data.frame(Sp.a, Sp.b, Sp.c,
     Sp.d, Sp.c)
7. >row.names(cockroach) <- c('BCI', 'LC',
     'FORT', 'BOQ', 'MIR', 'CORG')
```

## 11.1 PCA

```
>princomp(~VARIABLE1+VARIABLE2+VARIABLE3+ ...,
     cor=T)
```

Where the `princomp` factor is used to perform the PCA (centering and axis rotation) on the list of variables **VARIABLE1 + VARIABLE2 + VARIABLE3 +** .. supplied. The parameter **cor=T** specifies that a correlation (rather than covariance) matrix should be generated.

```
1. >cockroach.pca <- princomp(~ Sp.a + Sp.b
      + Sp.c + Sp.d + Sp.e, data=cockroach,
      cor=T)
2. >cockroach.pca <- princomp(cockroach,cor=T)
3. >cor(cockroach)
4. >cockroach.pca$sd^2
5. >screeplot(cockroach.pca, type='l')
6. >summary(cockroach.pca)
7. >biplot(cockroach.pca)
```

1. Use the `princomp` function to perform PCA on the list of variables from the **cockroach** data frame

2. An alternative, shorthand way of performing the same PCA as above when all of the variables in the **cockroach** data frame are to be included in the analysis

3. Generate a matrix of Pearson correlation coefficients between object pairs

4. List the latent Roots (Eigenvalues) for all the new principal components

5. Generate a scree plot. The argument `type='l'` specifies lines rather than bars for the graph.

6. List explained variance etc

7. Generate the ordination plot

## 11.2 Multidimensional scaling

```
>isoMDS(DIST)
```

Where the function `isoMDS` is used to perform Kruskal's non-metric multidimensional scaling on a matrix of distances (assumed to represent dissimilarities)

```
1. >library(vegan)
2. >cockroach.dist <- vegdist(cockroach,
      method='bray')
3. >cockroach.st <- decostand(cockroach, 2,
      method="max")
      OR
4. >cockroach.st <- wisconsin(cockroach)
5. >cockroach.dist <- vegdist(cockroach.st)
6. >library(MASS)
7. >cockroach.mds <- isoMDS(cockroach.dist,
      k=2)
8. >repeat{
9. >+cockroach.mds1<- isoMDS(cockroach.dist,
      initMDS(cockroach.dist), maxit=200,
      trace=FALSE, tol=1e-7)
10. >+if(cockroach.mds1$stress <
      cockroach.mds$stress) break
11. >+}
12. >par(mfrow=c(2,2))
13. >plot(procrustes(cockroach.mds1,
      cockroach.mds))
14. >plot(cockroach.mds1$points[,1],
      cockroach.mds$points[,1], type="n")
15. >text(cockroach.mds1$points[,1],
      cockroach.mds$points[,1],
      names(cockroach.dist))
16. >Sheppard(cockroach.dist,
      cockroach.mds1$points)
17. >par(mfrow=c(1,1))
```

1. Use the `library` function to load the **vegan** package if it has not already been loaded. This package contains a number of functions used in MDS

2. If no standardization is required, use the `vegdist` function to calculate a Bray-Curtis dissimilarity matrix and assign it to a `dist` object that is called **mds.dist**. The `method=..` parameter is used to specify the type of dissimilarity matrix (see the Table 15 for alternatives)

3. If standardization is required, use the `decostand` function. In the example, data are standardized by column (variable - species) maximas. The first parameter is a data frame (**(**mds), the second parameter specifies which margin (1=rows, 2=columns) to

standardize and the third specifies the form of standardization (see Table 14 for alternatives)

4. Use the `vegdist` function to calculate a Bray-Curtis dissimilarity matrix and assign it to a `dist` object that is called **mds.dist**. The `method=..` parameter is used to specify the type of dissimilarity matrix (see the Table 15 for alternatives)

5. Use the `library` function to load the **MASS** package if it has not already been loaded. This package contains the MDS functions

6. Use the `isoMDS` function to perform a Kruskal's non-metric MDS using the distance matrix (**mds.dist**) and specifying a 2-dimensional solution (with the parameter **k=2**). As no starting configuration is given, a classical solution is provided. The resulting MDS object is stored with the name **mds.mds**

7. Use the `repeat` procedure to recompute the MDS **200** times see if a better solution is possible (based on comparing stress values). This also demonstrates the use of the `initMDS` function which generates unique random starting configurations.

8. Setup the graphics window such that 4 graphs will be produced on the one page

9. Use the `plot` and `procrustes` functions to compare the the **cockroach.mds1** and **cockroach.mds** final configurations

10. Use the `plot` function to plot the final NMDS configuration of SU's without points (**type='n'**)

11. Use the `text` function to add the SU names as points on the NMDS configuration plot

12. Restore the graphics window to the default setting of a single graph per page

Table 14 Standardization methods available in the `decostand` function

| Standardization | Syntax |
|---|---|
| Divide by margin total | 'tot' |
| Divide by marginal maximum | 'max' |
| Average non-zero entry equals 1 | 'freq' |
| Unity of marginal sum of squares | 'normalize' |
| Range from 0 to 1 | 'range' |
| Zero mean and unit variance | 'standardize' |
| Presence/absence scale | 'pa' |

Table 15 Distance measures available in the `vegdist` function.

| Distance Measure | Syntax |
|---|---|
| Bray-Curtis | 'bray' |
| Euclidean Distance | 'euc' |
| City Block (Manhattan) | 'man' |
| Gower | 'gower' |
| Canberra | 'can' |
| Kulczynski | 'kul' |

## 11.3 Clustering

```
>hclust(DIST,METHOD)
```

Where the `hclust` function is used to calculate hierarchical clustering (and dendrogram generation) from the distance matrix (**D**IST using the linkage method (**METHOD**)

```
1. >cockroach.dist <- vegdist(cockroach)
2. >cockroach.clust <- hclust(cockroach.dist,
     method='ave')
3. >plot(as.dendrogram(cockroach.clust,
     edge.root=T, horiz=T))
4. >cockroach.coph <-cophenetic(
     cockroach.clust )
5. >cor(cockroach.dist,cockroach.coph)
```

1. Recycle the multivariate data set used for both the PCA and NMDS demonstrations. For the purpose of naming conventions copy the dissimilarity matrix (**cockroach.dist**) to a new name (**cockroach.dist**)

2. Use the `hclust` function to perform the hierarchical clustering from the **clust.dist** distance matrix using the **ave** (average or UPGMA) linkage method (see Table Linkages for alternative linkage methods)

3. Use the `plot` and `as.dendrogram` functions to produce a dendrogram from the hierarchical cluster object

4. Use the `cophenetic` function to calculate the cophenetic distances for the hierarchical clustering object (**cockroach.clust**)

5. Use the `cor` function to calculate the cophenetic correlation coefficient between the original distance measure **cockroach.dist** and the cophenetic distances **cockroach.coph**

Table 16 Linkage methods available in the `hclust` function.

| Linkage method | Syntax |
|---|---|
| Single (Nearest neighbor) | 'single' |
| Average (UPGMA) | 'ave' |
| Complete (furthest neighbor) | 'complete' |
| McQuitty | 'mcquitty' |
| Median | 'median' |
| Centroid | 'centroid' |
| Ward | 'ward' |

## 12 GRAPHS

### 12.1 Boxplots

Re-visiting the leaves (2 factor ANOVA) data set from section 9.4

Boxplots provide convenient means to examine the distributional and homeostoichastic assumptions associated with parametric analysis, examine the quality of the data and generally explore and visualize the data. Note however, that boxplots generated from less than 5 observations are of limited value.

```
>boxplot(VAR, ...)
OR
>boxplot(FORMULA, ...)
```

Where **VAR** is a single variable used to produce a single boxplot and **FORMULA** is a model statement (e.g. VARIABLE˜GROUPING) used to produce as many boxplots as there are levels within the **FORMULA** model

```
1. >boxplot(water ˜ temp, data=leaves)
2. >boxplot(water ˜ temp * humidity,
           data=leaves)
```

1. Use the `boxplot` function to plot boxplots for the variable (**water**) against **temp** in the **leaves** data frame.

2. Use the `boxplot` function to plot boxplots for the variable (**water**) against **temp**:**humidity** in the **leaves** data frame.

### 12.2 Bar graph

Re-visiting the leaves (2 factor ANOVA) data set from section 9.4

Presently there are no good bar graph routines in R. The command sequences suggested below provide a quick-fix approach to generating bar graphs. However, they fall short of being an elegant or even adequate solution.

```
>barplot(HEIGHTS, ...)
```

Where **HEIGHT** is an array of bar heights (usually means)

```
1. >leaves.split <- split(leaves$water,
     leaves$humidity:leaves$temp)
2. >leaves.m <- sapply(leaves.split,mean)
     OR
3. >leaves.m <- tapply(leaves$water,
     leaves$humidity:leaves$temp, mean)
4. >leaves.sd <- sapply(leaves.split,sd)
5. >leaves.n <- sapply(leaves.split,length)
6. >leaves.se <- leaves.sd/sqrt(leaves.n)
7. >leaves.bar <- barplot(leaves.m,
     col=c(0,1), ylim=c(0,max(leaves.m)*1.25))
8. >arrows(leaves.bar, leaves.m, leaves.bar,
     leaves.m+leaves.se, ang=90)
     OR
9. >library(gregmisc)
10. >plotCI(leaves.bar, leaves.m,
     uiw=leaves.se, add=T, gap=0)
```

1. Use the `split` function to split the values of **water** within the data set (**leaves**) according to the number of levels indicated by the second parameter (**leaves$humidity:leaves$temp**), which in this case is the interaction of the two factors. Store the split data set in the variable **leaves.split**. Note that when setting up for a bargraph with 2 factor variables, put the factor with the most levels first in the sequence (in this case humidity went first because it had 3 levels and temp only had 2).

2. Use the `sapply` function to calculate the mean of the **water** variable separately for each level in the

split data frame **leaves.split**

3. As an alternative the means could have been calculated from the original (unsplit) data frame (**leaves**) using the `taply` function

4. Use the `sapply` function to calculate the standard deviation of the **water** variable separately for each level in the split data frame **leaves.split**

5. Use the `sapply` function to calculate the length of the **water** variable separately for each level in the split data frame **leaves.split**

6. Use the standard deviations and sample sizes to calculate the standard error of the **water** variable separately for each level in the split data frame **leaves.split**

7. Use the `barplot` function to generate a bar graph using the means as the heights of the bars. The x-coordinates of each bar in the graph are stored in the variable **leaves.bar**. The parameter `ylim=c(0,max(leaves.m)*1.25)` is used to ensure that the y axis is high enough to accommodate error bars - the y-maximum has been set to 25% higher than the group with the highest mean

8. Use the `arrow` function to add upper error bars to each bar. The first two parameters used in this function (**leaves.bar** and **leaves.m**) are arrays of x and y coordinates used for the start of each line. These coordinates provided are for the middle-top of each bar (column) in the graph. The next two parameters (**dat.bar**, **dat.m+dat.se**) are arrays of x and y coordinates used for the end of each line. These coordinates provided are for the middle-top of each bar (column) plus the standard error in the graph. `ang=0` specifies that the head of the arrows should be at 90 degrees to the shaft.

9. Alternatively use the `plotCI` function to plot the error bars. The first two parameters used in this function (**leaves.bar** and **leaves.m**) are arrays of x and y coordinates used for the start of each error bar. The next parameter `uiw=`**`leaves.se`** indicates the height of the error bars. The parameter `add=T` specifies that the error bars (intervals) should be added to a current graph rather than replotting just the error bars.

## 12.3 Scatterplot

Revisiting the **snails** data set that was generated in section 8.2. This will be used to demonstrate simple scatterplots in R.

```
>scatterplot(FORMULA,smooth=F, boxplot=F)
```

Where **FORMULA** is a of the format
`dependent ~ independent | grouping`

```
1. >plot(species ~ salinity, dat=snails)
2. >abline(lm(species ~ salinity, dat=snails))
3. >lines(lowess(snails$salinity,
     snails$species, f=.5), col=2)
4. >scatterplot(species~salinity, smooth=T,
     reg.line=lm, data=snails)
```

1. Use the `plot` function to construct a scatterplot of of **species** on the y-axis and **salinity** on the x-axis
2. Use the `abline` (fits lines to plots) and `lm` (linear regression procedure) functions to fit a linear regression line through the data.
3. Use the `lines` (draws lines on plots) and `lowess` (performs lowess smoothing) functions to fit a red (`col=2`) lowess smoother through the data. The `f=.5` argument defines the smoother span
4. Use the `library` function to load the `car` package which contains the `scatterplot` function listed bellow
5. Use the `scatterplot` function to construct a scatterplot of **species** on the y-axis and **salinity** on the x-axis with a lowess smoother and a linear regression line fitted through the data and boxplots in the axes.

## 12.4 Interaction plots

Revisiting the 2 factor ANOVA **leaves** data set that was generated in section 9.4.

```
>interaction.plot(FACTOR1, FACTOR2, DV)
```

where **FACTOR1** represents the x-axis, the levels of the **FACTOR2** factor are used to define the separate traces and **DV** represents the y-axis

```
1. >interaction.plot(leaves$humidity,
     leaves$temp,leaves$water)
```

1. Use the `interaction.plot` function to generate an interaction plot with mean **water** on the y-axis, **humidity** on the x-axis and levels of **temp** to define the separate traces.

## *12.5 Saving graphs*

Graphs may saved for use in other documents.
Saving graphs in commands involves redirecting the output from a window (display) graphics device to a file filter graphics device. So when satisfied with the graph in the window display, redirect the output to either a JPEG or Encapsulated postscript device and resubmit the commands required to perform the graph. When the command sequence is finished, turn the device off so as to return control back to the standard output device (the screen). *Encapsulated postscript*

```
1. >postscript(file="filename.eps",
     paper="special", height=6, width=6,
     family="Helvetica")
2. >....
3. >dev.off()
```

1. Use the `postscript` function to produce a JPEG image. The parameter `file=` specifies an output file name (including full path otherwise path is assumed to be current working directory), `paper` specifies the size of the paper. While it is possible to specify **"a4"** as the paper size, for graphs that are intended to be embedded within other documents, it is better to use the value **"special"** and allow the `height` and `width` parameters specify the dimensions (in inches). The `family` parameter specifies the font family for any text on the graph. There are a large number of Adobe font names values possible (including **"Times"**, however **"Helvetica"** (similar to Arial) is preferable.
2. Re-enter the commands necessary to generate the graph
3. Turn off the postscript output graphics device, returning the control back to the standard output device (screen)

*JPEG*

```
1. >jpeg(filename="filename.jpg", height=500,
     width=500, quality=52)
2. >....
3. >dev.off()
```

1. Use the `jpeg` function to produce a JPEG image. The parameter `filename=` specifies an output file name (including full path otherwise path is assumed to be current working directory), and `height` and `width` specify the dimensions of the graphic in pixels. The `quality` parameter is used to specify the percentage of quality for compression (smaller values - greater compression, lower quality).
2. Re-enter the commands necessary to generate the graph
3. Turn off the jpeg output graphics device, returning the control back to the standard output device (screen)

# 13 Using command scripts

Series of commands can be written as plain text in any text editor, whereby each line represents a single R command. A collection of one or more commands is therefore a script. Scripts can be read into R using the `source` function.

```
source("filename")
```

Where **filename** is the file name of the script (including full path if not current working directory). Each line of the script (**filename**) will be parsed (checked for errors), interpreted, and run as if it had been typed directly at the > prompt.
This is an extremely useful feature as it enables complicated and/or lengthy sequences of commands to be stored, modified and reused rapidly as well as acting as a record of data analysis and a repository of analysis techniques.

**Murray Logan, School of Biological Sciences, Monash University.**
**February 9, 2007**