1

Very brief introduction to R

Learning objectives

1. Learning some of the basic R concepts: functions, objects, assigning, packages, mirrors, CRAN, and how to read data and access packages.
2. Statistical concepts reinforced are looking at data, transforming data, and there is detailed discussion of *skewness*.
3. We introduce you to the *bootstrap*, which will be used for several examples in this book.

R was developed as a free alternative to the powerful statistics language/program S/S-Plus. R and S-Plus are similar, and many of the procedures written in one will run in the other, but R is free and S-Plus is a commercial product. R is rapidly increasing in popularity. When statisticians develop procedures they often write R functions so that others can use them. Several books for learning to use R are listed at the end of this chapter.

R AND THE INTERNET

When using R it is useful to have an internet connection. Figure 1.1 shows a schematic of how the R system can be considered. From your computer you download R from the internet onto your computer so that you can later use the software without being on the internet. The program is available both from the R home page and from one of the CRAN (Comprehensive R Archive Network) mirror sites. Mirror sites, or mirrors, are sites that are supposed to be exactly the same as the main CRAN site. This means that when people download files they do not all have to do it from the same server. This makes downloading faster.

To begin using R you have to download it from one of the many R mirror sites: http://cran.r-project.org/mirrors.html. If an entire class is downloading information you should use different mirrors. Press the Windows, the Mac, or the Linux button in the 'precompiled binary distribution' box. Press 'base' and then run the setup program. Follow the wizard's instructions. This gives you both a very powerful statistics language and statistics package. This will allow you to do most of the statistics that you

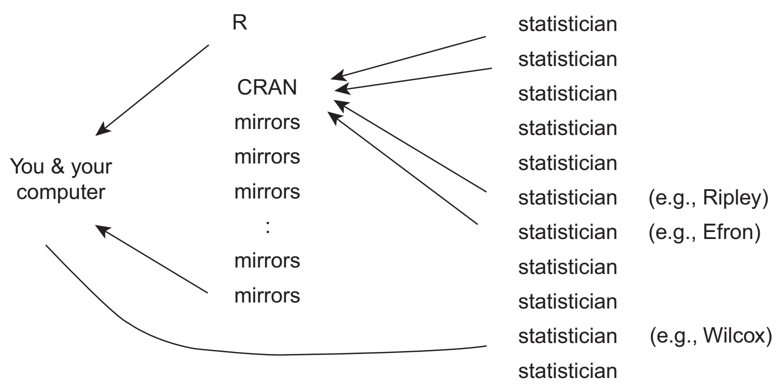


Figure 1.1 A schematic of the R-system

would want, but not all. Statisticians write their own packages for specialist purposes. Some submit these to CRAN so that others can use them. When a package is sent to CRAN it gets copied onto all the mirror sites. You can then download packages from there. For example, there is a package called **foreign** (R core members et al., 2007) that allows you to read data from other statistics programs directly into R. If you type:

```
install.packages("foreign")
```

a window like Figure 1.2 opens. We chose the server in Michigan (USA (MI)) since that is close to where we are preparing this chapter (in Ohio). This should install the package onto your computer. The package is now on your computer so you may access it in the future from this computer even if you are not connected to the internet assuming it is not erased. However, because authors update their packages frequently, it is worth reinstalling packages relatively regularly.

If the mirror is not perfectly up-to-date or if you are not connected to the internet it may not install. You may also get some warning messages.

Although the package is now on your computer (in a folder in the R directory) it is not active. To make it active type:

```
library("foreign")
```

Now you have access to a large number of functions that are used to import and export data between R and other statistical packages.

Some packages will have been installed when you downloaded R and you will just need to load these. You will need to download others from CRAN. Some statisticians, like Rand Wilcox, keep their packages on their own web page. In the case of Wilcox, he has written a book that effectively acts as both a manual and teaching resource for his functions. We will use some of his code in one of the examples and his code can be accessed from the web using the **source** function. We have only written ‘statistician’ on the right of Figure 1.1, but there are people from other disciplines (like computer science and psychology) who write packages for R. We did not include them because it is primarily statisticians doing the writing.

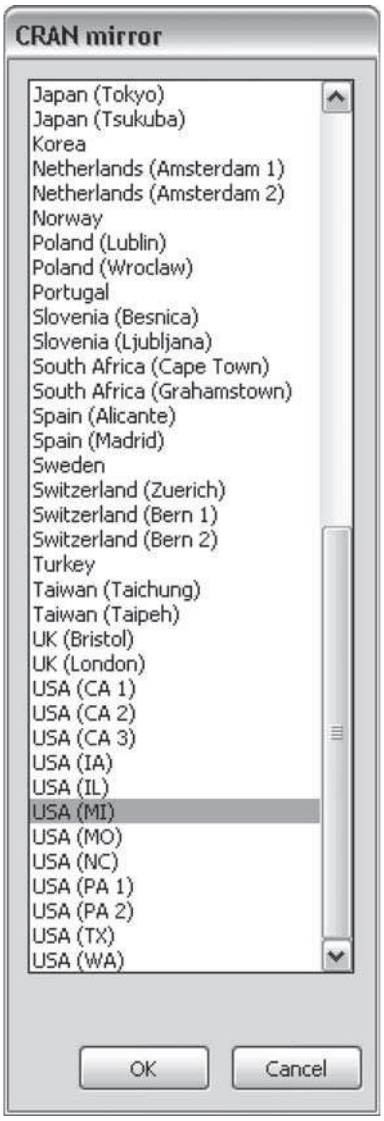


Figure 1.2 The window to choose a mirror site

Most of the introductory R books (a long list of them is on http://www.r-project.org/doc/bib/R-books.html; our favorite introduction is Crawley, 2005) go through how to use R like a calculator. Type `6+4`, or something like that, and see what happens. These books also go through the different data formats. A good (short and also free) book is available on http://cran.r-project.org/doc/manuals/R-intro.pdf.

If you have not already opened R, it would be good to open up it now because we will be telling you to type things in throughout the rest of this chapter.

FUNCTIONS AND OBJECTS

R works by applying *functions* to *objects*. To illustrate functions and objects we will show how to calculate the mean of four numbers. We will use the function **mean** to do this, but first we have to create a variable. A variable is a set of several similar objects. We can create a variable by *assigning* a list of values to a variable name. To make assignments you use the **<-** or **-** characters. So the following (and type this into R yourself) creates a variable **scores** that has four numbers (5, 6, 7 and 8).

```
scores <- c(5,6,7,8)
```

Another function, **c**, which stands for concatenate, is frequently used in R. This function tells R that these four numbers are a set. If we type:

```
scores
```

we get:

```
[1] 5 6 7 8
```

scores is a numeric variable, which is a type of object. R is case sensitive so typing **Scores** would have produced an error. An alternative way to write a sequence of numbers is with a colon **:**, such that

```
5:8
```

also produces the sequence

```
[1] 5 6 7 8
```

There is also a sequence function, **seq**, which can be used:

```
seq(5,8)
```

```
[1] 5 6 7 8
```

This function is useful if you want more complex sequences. If you type:

```
seq(10,30,5)
```

you get the sequence from 10 to 30 in steps of 5 units:

```
[1] 10 15 20 25 30
```

In R everything is an object, including variables. We can apply the function **mean** to this type of object (i.e., to numeric variables) by typing:

```
mean(scores)
```

and we get:

```
[1] 6.5
```

The [1] in front of 6.5 is because some functions produce several pieces of information so their parts are labeled. You can use functions in creating new variables. For example, you may want to have a variable for how far away each value is from the mean of the variable. The following command does this:

```
residscores <- scores - mean(scores)
```

Typing

```
residscores
```

produces

```
[1] -1.5 -0.5 0.5 1.5
```

Functions in R work with certain types of objects. While you can take a mean of four numbers, you cannot take a mean of four people's names. Names (or string values) need to be placed in quotation marks so that R does not think they are objects that it should be able to find. The following creates a variable of four people's names and shows that the function **mean** does not work in this instance.

```
Simpsons <- c("Homer", "Marge", "Lisa", "Bart")
mean(Simpsons)

[1] NA
Warning message:
argument is not numeric or logical: returning NA in:
mean.default(Simpsons)
```

The function **c** works also with strings. Thus, when Maggie arrived at the Simpson household, the glorious event of childbirth could be written in R as:

```
Simpsons <- c(Simpsons,"Maggie")
```

We can identify each member of the Simpsons by writing the variable name with an index. So, to identify the fourth member of the Simpsons we write:

```
Simpsons[4]

[1] "Bart"
```

When a function is applied to an object it can create another object, which can then be used in other functions. The basic regression function is **lm**. It is applied to some data objects (a response variable and some predictor variables) and a **lm.object** is created. This object can then be used in other functions, like **plot**, **summary**, and **anova**.

These are illustrated in several examples throughout this book. Many R functions are intelligent. For example, the **`plot`** function works differently dependent upon what type of object is placed within it, and we will see this as we go through this book.

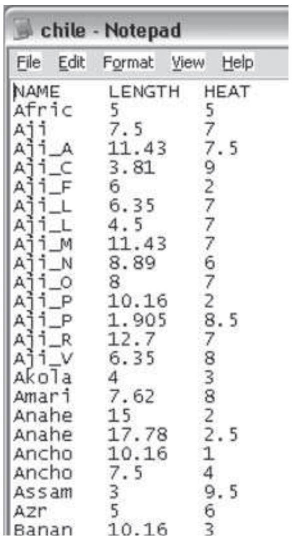
Example 1 – Chile heat and length

- Data: Downloaded from www.pepperjoe.com, May 2006
- Research question: Are the length and heat of a chile related?
- Purpose: Showing how to load data from a text file and from SPSS, how to access packages, and run some preliminary statistics and graphs. The statistical concepts are skewness, transformations, and simple linear regression.

The basic methods in R to read data assume that the data are in a text file in a table-like form. Figure 1.3 shows data in this format in a Notepad file. It is important to have data in a text (or ASCII) format, so if your data are in a word processing file you have to use the SAVE AS option. If the data are in text format and in a table like Figure 1.3, then the command:

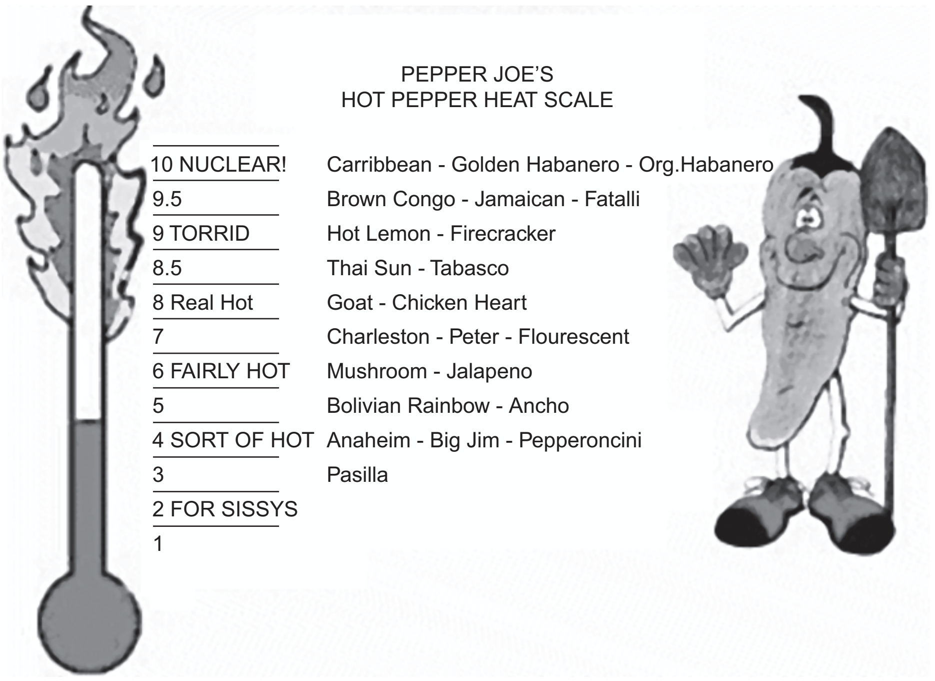
```
newdata <- read.table("filename")
```

assigns the data in "filename" into the object **`newdata`**. When using **`read.table`** the **`"filename"`** may either be a file on your computer or one on the web. For the function used in this example, **`read.spss`**, the data must be saved on your computer. The characters **`<-`** assign things to each other, and can be used in either direction so **`x <- 6+4`** and **`6+4 -> x`** both assign the value 10 to **`x`**. Often you will have data in another package and need to import it into R. The package **`foreign`** allows data to be read from other packages, including: SPSS, Minitab, SYSTAT, SAS, and Stata. Since it seems SPSS is the most popular among academic psychologists we will assume that you want to import the data from SPSS.



```
NAME    LENGTH  HEAT
Afric   5       5
Aji     7.5     7
Aji_A   11.43   7.5
Aji_C   3.81    9
Aji_F   6       2
Aji_L   6.35    7
Aji_L   4.5     7
Aji_M   11.43   7
Aji_N   8.89    6
Aji_O   8       7
Aji_P   10.16   2
Aji_P   1.905   8.5
Aji_R   12.7    7
Aji_V   6.35    8
Akola   4       3
Amari   7.62    8
Anahe   15      2
Anahe   17.78   2.5
Ancho   10.16   1
Ancho   7.5     4
Assam   3       9.5
Azr     5       6
Banan   10.16   3
```

Figure 1.3 A text file for the chile data set as shown within Notepad



10 NUCLEAR!	Carribbean - Golden Habanero - Org.Habanero
9.5	Brown Congo - Jamaican - Fatalli
9 TORRID	Hot Lemon - Firecracker
8.5	Thai Sun - Tabasco
8 Real Hot	Goat - Chicken Heart
7	Charleston - Peter - Flourescent
6 FAIRLY HOT	Mushroom - Jalapeno
5	Bolivian Rainbow - Ancho
4 SORT OF HOT	Anaheim - Big Jim - Pepperoncini
3	Pasilla
2 FOR SISSYS	
1	

ED: Please provide citation for figure 1.4

Figure 1.4 The heat scale. from www.pepperjoe.com/about/heatscale.html

The data in this example examine the relationship between the length of a chile and its heat. **LENGTH** is in cm and **HEAT** is in Pepper Joe's Hot Pepper Heat Scale (in technical papers the Scoville scale is used, but they don't have a smiling chile with a shovel waving at you, see www.pepperjoe.com/about/heatscale.html).

The **library** function loads the package **foreign** so that it can be used. Note that \\ are used rather than \ in the **read.spss** function (and the **read.table** function used below). The **attach** command makes the data file the active data file overwriting any other variables that may have the same names. The following commands read data from c:\temp\chile.sav. The data are available on the book's web page (on www.sagepub.co.uk/wrightandlondon) so this file will need to be copied into your c:\temp folder (if you do not have a c:\temp folder copy it elsewhere and change the code below accordingly).

```
library("foreign")
chile <- read.spss("c:\\temp\\chile.sav")
attach(chile)
```

You may get a warning, but this particular warning should not a problem; these data are read accurately. You do not need to have SPSS on your computer to read SPSS data files into R.

The data are also stored on this book's web page as a text file. They can be accessed by:

```
chile <-
read.table("http://www.sagepub.co.uk//wrightandlondon//
   chile.dat",header=T)
```

This command takes up multiple lines to print and requires that you very carefully write the web page each time. So that you do not have to write a command that is multiple lines, it is worth assigning this book's web page to the object **webreg** so that it need not be typed each time.

```
webreg <- "http://www.sagepub.co.uk//wrightandlondon//"
```

To write the full web address we have to **paste** together the web page and the file name.

```
paste(webreg,"chile.dat",sep="")
```

```
[1] "http://www.sagepub.co.uk//wrightandlondon//chile.sav"
```

The default for the **paste** function is to have one space between each of the objects that are pasted together. Having a space in a web address will cause the web address not to be recognized, so we have to tell R that there should be no separation between the objects. The option **sep=""** tells the computer this. If we wrote **sep=","** it would have placed a comma between each part.

Now we can write:

```
chile <- read.table(paste(webreg,"chile.dat",sep=""),
   header=T)
```

The **header=T** means that the first line in the data file has the variable names (see Figure 1.3). The **T** stands for **TRUE** and it can be written as **header=TRUE**. Alternative methods for reading data from the book's web page are listed in the box below. If you plan to use this book while disconnected from the internet, there are instructions on the web page for downloading all the data, functions, and syntax to your hard drive (or memory stick).

There are many ways to access data from the book's web page and three are presented here. The first is to write out the web page within the **read.table** command:

```
chile <-
read.table("http://www.sagepub.co.uk//wrightandlongon//
   chile.dat", header=T)
```

The problem is that this takes a few lines and requires careful typing.

The second is to first assign the web page to an object, like **webreg**, and then use the shorter command.

```
webreg <- "http://www.sagepub.co.uk//wrightandlongon//"
chile <- read.table(paste(webreg,"chile.dat",sep=""),
   header=T)
```

(*Cont'd*)

This still requires assigning the web address to **webreg** every time you turn on R and want to access the book's web page, but at least the first line can be copied and pasted from other exercises.

The final option, which will make life easier for people who are fairly comfortable with computers, is to add the line

```
webreg <- "http://www.sagepub.co.uk//wrightandlondon//"
```

into the file called Rprofile.site which was installed when you installed R. Use the search facility on your computer to find it. If you add this line then everytime you start R this assignment will be made. Then you would only need to type:

```
chile <- read.table(paste(webreg,"chile.dat",sep=""),
   header=T)
```

Because this final option requires more computer knowledge than we are assuming, the second option will be used throughout this book.

After you import the data set (and name it, say **chile**), then type:

```
attach(chile)
```

This attaches the data set to your working environment. 'Attaching to your working environment' is the technical jargon that means you can now access all these variables by just typing the variable names. To find out the variable names type:

```
names(chile)
```

```
[1] "NAME" "LENGTH" "HEAT"
```

These are the same as in Figure 1.3. While you can access a variable without the file being attached by typing:

```
chile$HEAT
```

this can get cumbersome and is not useful if you are working with only one data set at a time. It is easier not having to re-type the name of the data set each time. Therefore, in this book we will always **attach** the data set we are working with. If you are doing more advanced statistics where you are looking at several data sets, we recommend not attaching the data but accessing them within each function or using the **with** function (see Crawley, 2007). However, for most psychologists' needs this would add further complications.

When R reads SPSS files, all the variable names are UPPERCASE. This is how SPSS stores them internally because SPSS is not case sensitive. R is case sensitive so:

```
heat
```

produces the word NULL. This means there is no variable **heat**. The variable is **HEAT**.

If you want to find out how many cases are in the variable **LENGTH**, you need to ask for the **length** of **LENGTH**:

```
length(LENGTH)
```

length is an R function, **LENGTH** is a variable in the chile dataset. Be careful not to assign **length <- LENGTH**, because, while R will try to figure out what you want, this can lead to errors.

We are now going to describe *skewness* in more depth than is typical in introductory statistics courses. We do this for two reasons. First, it emphasizes the importance of looking at your variables and describing their distributions. This is something that the American Psychological Association states should be done (Wilkinson et al., 1999). Second, it provides a good way to focus on how well different transformations work. Statisticians stress the importance of transformations (e.g., Mosteller & Tukey, 1977), but this stress is often absent from psychology texts.

We now look at the histogram of this variable. This can be done with **hist(LENGTH)** and the result is shown in the left panel of Figure 1.5. It looks fairly skewed. There are several different definitions of skewness (Groeneveld & Meeden, 1984) and none are in the base package R, so you have to load another package with this command, the two most commonly used being **e1071** (Dimitriadou et al., 2006) and **fBasics** (Harrell et al., 2005). This loads a function, **skewness**, which is the most common measure of skewness.[1] It is:

$$skewness = \frac{\sum (x_i - \bar{x})^3}{n(sd(x)^3)}$$

[1]The **skewness** function in both of these R packages provides the value for the sample skewness, not the estimate of the population skewness. In your introductory statistics textbooks there may have been some discussion of the difference between these with respect to the standard deviation. A better estimate for the population skewness of a variable **x** is, in R code:

```
n <- length(x)
popskew <- sqrt(n*(n-1))/(n-2)*skewness(x)
```

In standard equations, this is:

$$est.\ pop.\ skew = \frac{\sqrt{n(n-1)}}{(n-2)} sample\ skew$$

As with the standard deviation, the difference is usually only slight, so people tend not to worry about this. For the variable **LENGTH** the population estimate is 1.196.

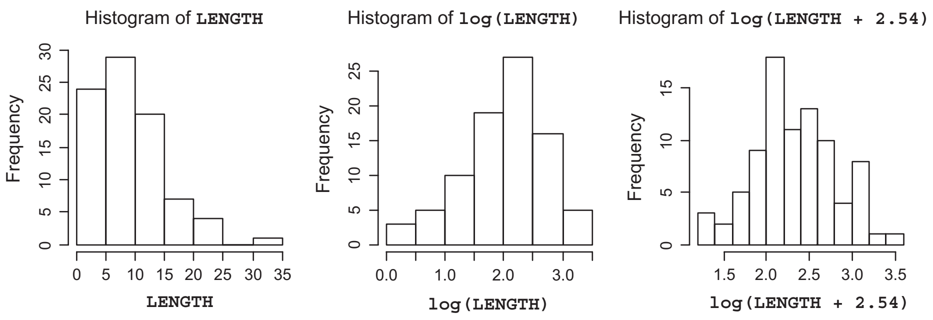


Figure 1.5 The histograms for the variable **LENGTH**, in the left panel, the transformed variable **log(LENGTH)**, in the middle panel, and transformed variable **log(LENGTH + 2.54)**, in the right panel

We will use **e1071**:

```
install.packages("e1071")
library(e1071)
skewness(LENGTH)

[1] 1.174819
```

which is a pretty high skew (anything over 1 is usually considered high). An equation for standard error of skewness assuming the data are normally distributed is:

$$se_{skewness} = \sqrt{\frac{6(n-2)}{(n+1)(n+3)}}$$

This produces 0.256 for these data. Sometimes an approximation, $\sqrt{6/n}$, is given, and this yields 0.266, so very similar. This means an approximate 95% confidence interval for these data is: $1.17 \pm 2(.26) = 1.17 \pm .52 = (0.65, 1.69)$. One of the problems with this approach is the assumption of normality when it is likely that you are interested in distributions which are not normal.

A *bootstrap sample* is one taken from the observed sample where you randomly choose one item, record its value, and then return this item to the sample. You then randomly choose a second item, record, and return, and repeat this until you have a sample as large as the original sample. Some items will be chosen multiple times because each time an item is randomly chosen. You can then calculate whatever statistic you want for this bootstrap sample (i.e., the mean, skewness, the F from an ANOVA).

Using a computer you can repeat this procedure thousands of times and record the relevant statistics for each bootstrap sample. The distribution of the statistics

(*Cont'd*)

for these bootstrap samples provides a way of measuring the precision of your estimates. A rough way to esti mate the 95% confidence interval of any statistic is the middle 95% of the distribution for these bootstrap samples. Statisticians have devised ways to improve upon this rough approximation to estimate the confidence intervals and the bias-corrected and accelerated (BCa) method is used here.

Bootstrapping is a very flexible procedure and is rapidly gaining in popularity. It can work on many problems where the traditional mathematical approach has difficulties. Bootstrapping supplements the mathematical approach to statistics with the brut computing force of being able to create thousands of bootstrap samples in seconds!

In these situations a bootstrap estimate can be useful (for a 'leisurely' introduction see Efron & Gong, 1983). We discuss bootstrapping more throughout this book and in the box above. Bootstrapping runs the test over and over by resampling values from those observed data (with replacement). A distribution of the relevant statistic, here skewness, is created. The middle 95% of them is a rough estimate of the 95% confidence interval. More advanced techniques create what are generally thought of as better intervals, and at the time of writing the bias-corrected and accelerated, or BCa, intervals are preferred. We use the package **`boot`** (Canty & Ripley, 2008). This package is part of CRAN and is automatically installed with R. The **`boot`** function requires the following: first the variable or variables that you will be using, then the function which you wish to apply to that variable, and then the number of bootstrap samples (here **`R=1000`**). The complex part is writing the function. For skewness you need to say **`function(x,i) skewness(x[i])`**. The **`x`** stands for the variable **`LENGTH`** and the **`i`** is for the 1000 replications. When there are multiple variables and more complex functions this can become more difficult, but we will address this in later chapters. Running the function **`boot`** creates a bootstrap object that can be placed within other functions.

One of these functions, **`boot.ci`**, produces the most commonly used confidence limits. Differences between these intervals are described in DiCiccio & Efron (1996) and other reviews of bootstrapping. We just concentrate on the BCa estimates. If you had just wanted to print these you could have typed: **`boot.ci(lengthboot, type="bca")`**.

```
library(boot)
lengthboot <- boot(LENGTH,function(x,i) skewness(x[i]), R=1000)
boot.ci(lengthboot)
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = lengthboot)
```

```
Intervals :
Level       Normal              Basic
95%    ( 0.644, 1.826 )    ( 0.627, 1.765 )
Level      Percentile           BCa
95%    ( 0.584, 1.723 )    ( 0.714, 1.962 )
Calculations and Intervals on Original Scale
Some BCa intervals may be unstable
```

Whichever estimates are used, this variable is skewed (skewness is zero for perfectly symmetric data). Because of this skewness we might want to transform the data; the natural logarithm (**log**) is a popular choice for positively skewed data. The middle panel of Figure 1.5 shows the distribution for this transformed variable is more symmetrical than the untransformed variables (in the left panel), but now its tail seems drawn out to the left. It appears negatively skewed and the statistics confirm this.

```
lnlength <- log(LENGTH)
skewness(lnlength)
```

```
[1] -0.4703981
```

The 95% confidence interval just overlaps with zero if using the traditional ±2se approach and intervals that do not overlap with zero with the bootstrap method. Because bootstrap estimates depend on the particular bootstrap samples chosen, if you run a bootstrap confidence interval several times you may find some of them do overlap (but most will not). If we add an inch to each chile (or 2.54 cm) and then log the data, the skewness is near zero (**skewness(log(LENGTH+2.54))** produces 0.04). This type of transformation is described in the classic regression book by Mosteller & Tukey (1977), where the 2.54 is called a starting value. It is also referred to as a flattening constant, delta, and a Bayesian flat prior depending on the situation in which it is used. Figure 1.5 shows the three histograms next to each other. To tell the computer to print multiple graphs on the same page you have to use the **par(mfrow=c(1,3))** command. This is a tricky command to remember, but it is often used so is worth memorizing. This tells R that the graph window should have 1 row of graphs and 3 columns of graphs. If you had wanted the graphs in a 2 x 2 grid, you would have typed: **par(mfrow=c(2,2))**. When you are done with any of these multiple-graph-figures it is worth returning this to its default **par(mfrow=c(1,1))**, which has just one graph per figure. You will need to re-shape the graph window to make the figures look like those that are printed in this book. If you right-click on the window you can copy it as a metafile or a bitmap and paste it into other documents in packages including Word and PowerPoint. The following code makes Figure 1.5.

```
par(mfrow=c(1,3))
hist(LENGTH)
hist(log(LENGTH))
hist(log(LENGTH+2.54))
par(mfrow=c(1,1))
```

To end a session, you should detach the data set, here `detach(chile)`. To quit, type `q()`. This is one of the few R functions that has no input into it.

This was an incredibly brief introduction to a very powerful statistics environment. As shown by the rapidly increasing list of R books on http://www.r-project.org/doc/bib/R-books.html, this free and flexible environment is growing in popularity.

SOME WORDS/CONCEPTS WORTH REMEMBERING

R concepts

- functions: the verbs of R that are applied to objects;
- objects: everything in R;
- CRAN: Comprehensive R Archive Network;
- mirrors: where to download R stuff from;
- packages: sets of functions written for users.

R functions

- `read.spss` and `read.table`: to read SPSS and text files;
- `install.packages`: to access packages from CRAN;
- `library`: to make packages active;
- `seq` and `:`: to make sequences of numbers;
- `par(mfrow=c(7,4))`: to print with 7 rows and 4 columns of graphs;
- `log`: the logarithm function;
- `<-`: to assign objects to each other;
- `c`: to concatenate (or combine) objects.

Statistical concepts

- skewness: a measure of a distribution's symmetry;
- bootstrap: a modern method for estimating the precision of almost anything.

FURTHER READING

Crawley, M. J. (2005). *Statistics: An introduction using R*. Chichester, UK: Wiley. This is a great introduction. Although not written for psychologists, it is still excellent and very clear. Professor Crawley is actually a plant ecologist looking at the interactions between plants and animals, and has books on things like the 'Flora of Berkshire'. http://www3.imperial.ac.uk/naturalsciences/research/statisticsusingr

Crawley, M. J. (2007). *The R book*. Chichester, UK: Wiley. 950 pages of R and it's legal! This is part reference part teaching book. It is more advanced than his 2005 book, not in terms of statistical knowledge, but in terms of computing skills. http://www.bio.ic.ac.uk/research/mjcraw/therbook/index.htm

Fox, J. & Anderson, R. (2005). Using the R statistical computing environment to teach social science statistics courses. http://socserv.mcmaster.ca/jfox/Teaching-with-R.pdf and see also http://socserv.mcmaster.ca/jfox/Courses/R-course/index.html. John Fox has written much about R and statistics.

Venables, Smith and the R Development Core Team (2008). *An introduction to R*. Free on http://cran.r-project.org/doc/manuals/R-intro.pdf.
An Amazon list on learning R:
http://www.amazon.com/Learn-the-R-statistics-software/lm/244T3243F9I31/ref=cm_lmt_srch_f_2_rsrsrs0/102-3396071-7592139
The R team recommends various packages for different areas. See:
http://cran.r-project.org/src/contrib/Views/
http://cran.r-project.org/src/contrib/Views/SocialSciences.html

The R help facilities do not tell you about the statistical issues, just how to run the functions. Much R information can be downloaded from the R web site.

The code in this book was run using R 2.4–2.7, but we will update the code if it stops working (if someone tells us!). It is best to use the most up-to-date non-beta version if you are a non-expert.

Note: If using a word processor to type commands, and then pasting them into R, be careful that the symbols you type are not being changed. For example, some word processors might change <- to a single character for an arrow, ←, which R will not understand. This facility can be turned off (in Word, from the Tools/AutoCorrect toolbar). Other problem symbols are ”, ’, and ellipses (three dots). The line breaks are also sometimes not copied correctly. WordPad and Notepad, which have fewer auto-correcting procedures, are often better to use than the more encompassing word processing packages like Word and WordPerfect. Alternatively, you can use text editors designed for R like Tinn-R (http://www.sciviews.org/Tinn-R/ and http://sourceforge.net/projects/tinn-r).