Chapter 3
Accessing Variables and Managing Subsets of Data

In the previous chapter we demonstrated importing data from a spreadsheet or database into R. We also showed how to type in small datasets and store them in a data frame. We now discuss accessing subsets of the data.

3.1 Accessing Variables from a Data Frame

Assuming that no errors were encountered when importing the squid data in the previous section, we can now move on to working with the data.

During statistical analysis it can be important to remove portions of the data, select certain subsets, or sort them. Most of these operations can be done in Excel or other spreadsheet (or database) program prior to importing into R, but, for various reasons, it is best not to do this. You may end up needing to reimport your data each time you make a subselection. Also some data files may be too large to import from a spreadsheet. Hence, a certain degree of knowledge of manipulating data files in R is useful. However, the reader is warned that this may be the most difficult aspect of R, but once mastered it is rewarding, as it means that all the tedious data manipulation in Excel (or any other spreadsheet) can be done in R.

We use the squid data imported in the previous chapter. If you have not already done this, use the following commands to import the data and store it in the data frame `Squid`.

```
> setwd("C:/RBook/")
> Squid <- read.table(file = "squid.txt",
                      header = TRUE)
```

The `read.table` function produces a data frame, and, because most functions in R work with data frames, we prefer it over the `scan` function. We advise using the `names` command immediately after the `read.table` command to see the variables we are dealing with:

A.F. Zuur et al., *A Beginner's Guide to R*, Use R,
DOI 10.1007/978-0-387-93837-0_3, © Springer Science+Business Media, LLC 2009

```
> names(Squid)
[1] "Sample" "Year" "Month" "Location" "Sex" "GSI"
```

We often notice that our course participants continue typing code directly into the R console. As mentioned in Chapter 1, we strongly recommend typing commands into a good text editor, such as Tinn-R on Windows operating systems. (See Chapter 1 for sources of editors using non-Windows operating systems.) To emphasise this, Fig. 3.1 shows a snapshot of our R code so far. Note that we copied and pasted the results of the `names` command back into our Tinn-R file. This allows us to see quickly which variables we will work with, and reduces the chance of typing errors.

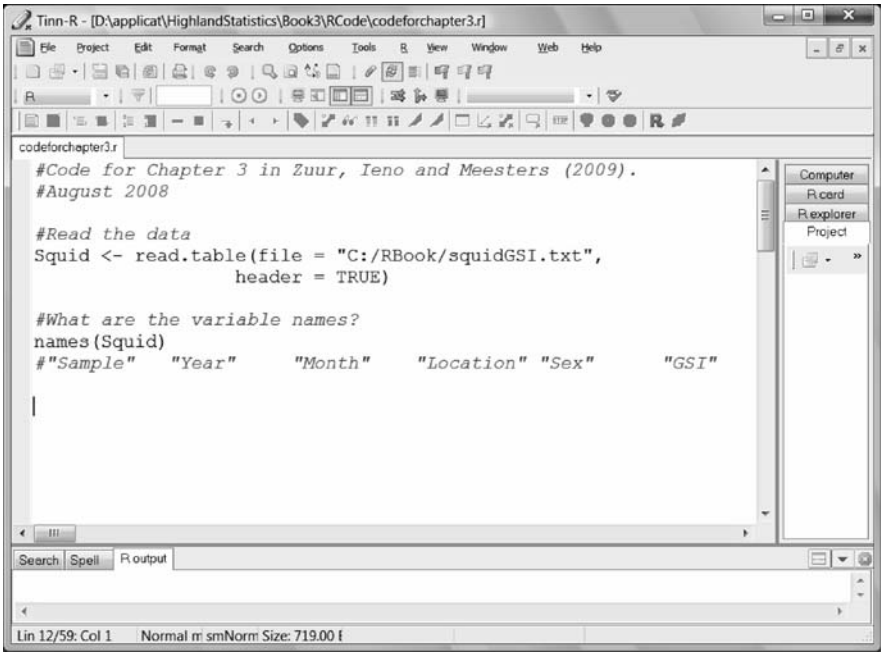

Fig. 3.1 Snapshot of our Tinn-R file. Note that the "#" symbol is put before comments, and that the code is well documented, including the date when it was written. Copying and pasting all variable names into the text file allows us to quickly check the spelling of the variable names. It is important that you structure your file as transparently as possible and add comments. Also ensure that you have a backup of this file and the data file

3.1.1 The str Function

The `str` (structure) command informs us of the status of each variable in a data frame:

```
> str(Squid)
'data.frame': 2644 obs. of 6 variables:
 $ Sample    : int 1 2 3 4 5 6 7 8 9 10 ...
 $ Year      : int 1 1 1 1 1 1 1 1 1 1 ...
 $ Month     : int 1 1 1 1 1 1 1 1 1 2 ...
 $ Location  : int 1 3 1 1 1 1 1 3 3 1 ...
 $ Sex       : int 2 2 2 2 2 2 2 2 2 2 ...
 $ GSI       : num 10.44 9.83 9.74 9.31 8.99 ...
```

This cryptic output tells us that the variables Sample, Year, Month, Location, and Sex are integers, and GSI is numeric. Suppose that you made a mistake with the point separation:

```
> setwd("C:/RBook/")
> Squid2 <- read.table(file = "squidGSI.txt",
                       dec = ",", header = TRUE)
```

We (wrongly) told R that the decimal separation in the ascii file is a comma. The `Squid2` data frame still contains the same data, but using the `str` command allows us to detect a major problem:

```
> str(Squid2)
'data.frame': 2644 obs. of 6 variables:
 $ Sample    : int 1 2 3 4 5 6 7 8 9 10 ...
 $ Year      : int 1 1 1 1 1 1 1 1 1 1 ...
 $ Month     : int 1 1 1 1 1 1 1 1 1 2 ...
 $ Location  : int 1 3 1 1 1 1 1 3 3 1 ...
 $ Sex       : int 2 2 2 2 2 2 2 2 2 2 ...
 $ GSI       : Factor w/ 2472 levels "0.0064","0.007" ...
```

The GSI variable is now considered to be a factor. This means that if, in continuing, we use functions such as the mean or a boxplot, R will produce cryptic error messages, as GSI is not numerical. We have seen a lot of confusion due to this type of mistake.

Therefore we strongly recommend that you always combine the `read.table` function with the `names` and `str` functions.

The variable of interest is GSI, and, in any follow-up statistical analysis, we may want to model it as a function of year, month, location, and sex. Before doing any statistical analysis, one should always visualise the data (i.e., make plots). Useful tools are boxplots, Cleveland dotplots, scatterplots, pair plots, and the like (see Zuur et al., 2007; 2009). However, R does not recognize the variable GSI (or any of the other variables). To demonstrate this, type

```
> GSI
Error: object "GSI" not found
```

The problem is that the variable `GSI` is stored in the data frame `Squid`. There are several ways to access it, both good ways and bad ways, and we discuss them next.

3.1.2 The Data Argument in a Function

The most efficient method of accessing variables in a data frame is as follows. Identify a function in R, for instance, `lm` for linear regression; specify the model in terms of the variables `GSI`, `Month`, `Year`, and `Location`; and tell the function `lm` that the data can be found in the data frame `Squid`. Although we are not further discussing linear regression in this book, the code would look as follows.

```
> M1 <- lm(GSI ~ factor(Location) + factor(Year),
           data = Squid)
```

We ignore the first part, which specifies the actual linear regression model. It is the last part of the statement (`data =`) which tells R that the variables are in the data frame `Squid`. This is a neat approach, as there is no need to define variables outside the data frame; everything is nicely stored in a data frame `Squid`. The major problem with this approach is that not all functions support the `data` option. For example,

```
> mean(GSI, data = Squid)
```

gives an error message:

```
Error in mean (GSI, data = Squid) : object "GSI" not
found
```

because the function `mean` does not have a `data` argument. And sometimes a help file tells you that there is a `data` argument, and it may work in some cases, but not in other cases. For example, the code below gives a boxplot (not shown here).

```
> boxplot(GSI ~ factor(Location), data = Squid)
```

But this command gives an error message:

```
> boxplot(GSI, data = Squid)
Error in boxplot(GSI, data = Squid) : object "GSI" not
found
```

To summarise, if a function has a `data` argument, use it; it is the neatest programming approach.

3.1.3 The $ Sign

So, what can you do if a function does not have a data argument? There are two ways to access a variable. The first option is the `$` sign:

```
> Squid$GSI
 [1] 10.4432  9.8331  9.7356  9.3107  8.9926
 [6]  8.7707  8.2576  7.4045  7.2156  6.8372
[11]  6.3882  6.3672  6.2998  6.0726  5.8395
```

< Cut to reduce space >

We only copied and pasted the first few lines of the output as the dataset contains 2644 observations. The other variables can be accessed in the same way. Type the name of the data frame followed by a `$` and the name of the variable. In principle, you can put spaces between the `$` sign and the variable names:

```
> Squid$GSI
 [1] 10.4432 9.8331 9.7356 9.3107 8.9926
 [6] 8.7707 8.2576 7.4045 7.2156 6.8372
[11] 6.3882 6.3672 6.2998 6.0726 5.8395
```

< Cut to reduce space >

We do not recommend this (it looks odd).

The second approach is to select the sixth column if you want to access the GSI data:

```
> Squid[, 6]
 [1] 10.4432 9.8331 9.7356 9.3107 8.9926
 [6] 8.7707 8.2576 7.4045 7.2156 6.8372
[11] 6.3882 6.3672 6.2998 6.0726 5.8395
```

< Cut to reduce space >

It gives exactly the same result. Using either `Squid$GSI` or `Squid[,6]`, you can now calculate the mean:

```
> mean(Squid$GSI)
[1] 2.187034
```

Our preference is the coding with `$GSI`. A week after you typed `Squid[, 6]`, you will have forgotten that the GSI data are in the sixth column, and the notation `$GSI` is clearer.

To add to the confusion, you can also use `Squid[,"GSI"]`. Note that in some functions, the use of the `Squid$` approach gives an error message, for example, the `gls` function from the `nlme` package.

3.1.4 *The* `attach` *Function*

Let us now discuss a bad way of accessing variables. We have used `"$"` to access variables from the data frame `Squid`. It can be tedious typing `Squid$` each time we want to use certain variables from the GSI dataset. It is possible to avoid this by using the `attach` command. This command makes all variables inside the data frame `Squid` available. To be more precise, the `attach` command adds `Squid` to the search path of R. As a result, you can now type `GSI` or `Location` without using `Squid$`.

```
> attach(Squid)
> GSI
 [1] 10.4432  9.8331  9.7356  9.3107  8.9926
 [6]  8.7707  8.2576  7.4045  7.2156  6.8372
[11]  6.3882  6.3672  6.2998  6.0726  5.8395
```

< Cut to reduce space >

The same holds for the other variables. As a result, you can now use each function without a `data` argument:

```
> boxplot(GSI) #Graph not shown here
> mean(GSI)
[1] 2.187034
```

The `attach` command sounds too good to be true. It can be a useful command, if used with great care. Problems occur if you attach a dataset that has variable names that also exist outside the data frame. Or if you attach two data frames, and variables with the same names appear in both. Another problem may occur if you attach a dataset that has variable names that match some of R's own function names or names of variables in example datasets (e.g., the variable name "time" and the function "time"). In all these cases, you may discover that R will not use the variable in your calculations as you expected. This is a major problem in classroom teaching when students do different exercises and each time load a new dataset with similar names such as "Location," "Month," "Sex," and so on. In such situations it is better to use the `detach` command, or simply close and restart R each time you work with a new dataset. If you use only one dataset for a research project and are careful with variable names, the `attach` command is extremely useful. But use it with care.

To summarise the use of the `attach` command,

1. To avoid duplicate variables, do not type the `attach (Squid)` command twice.
2. If you use the `attach` command, make sure that you use unique variable names. Refrain from common names such as Month, Location, and the like.
3. If you import multiple datasets, and only work with one dataset at a time, consider removing a data frame from the search path of R with the `detach` command.

In the remaining sections of this chapter, we assume that you did not type in the `attach(Squid)` command. If you did, type

```
> detach(Squid)
```

Do Exercise 1 in Section 3.7. This is an exercise in using the `read.table` function and accessing variables using an epidemiological dataset.

3.2 Accessing Subsets of Data

In this section, we discuss how to access and extract components of the data frame `Squid`. The methods can be applied on a data frame that you created yourself by typing in data, as in Chapter 2.

The situation may arise in which you only want to work with, for example, the female data, data from a certain location, or data from the females of a certain location. To extract the subsets of data, we need to know how sex was coded. We could type in

```
> Squid$Sex
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[23] 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[45] 2 2 2 1 2 2 2 2 2 2 2 2 2 1 2 1 1 1 1 2 1 1
[67] 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1
```

< *Cut to reduce space* >

but this shows all values in the variable `Sex`. A better option is to use the `unique` command that shows how many unique values there are in this variable:

```
> unique(Squid$Sex)
[1] 2 1
```

The 1 stands for male, and the 2 for female. To access all the male data, use

```
> Sel <- Squid$Sex == 1
> SquidM <- Squid[Sel, ]
> SquidM
     Sample Year Month Location Sex    GSI
24       24    1     5        1   1 5.2970
48       48    1     5        3   1 4.2968
58       58    1     6        1   1 3.5008
60       60    1     6        1   1 3.2487
61       61    1     6        1   1 3.2304
```

< *Cut to reduce space* >

The first line creates a vector `Sel` that has the same length as the variable `Sex`, with values that are TRUE if `Sex` equals 1, and FALSE otherwise. Such a vector is also called a Boolean vector, and can be used to select rows, hence our name `Sel`. On the next line, we select the rows of `Squid` for which `Sel` equals `TRUE`, and we store the selected data in `SquidM`. Because we are selecting rows of `Squid`, we need to use the square brackets [ ], and, as we want rows, the vector `Sel` with Boolean values must go *before* the comma. It is also possible to do both lines in one command:

```
> SquidM <- Squid[Squid$Sex == 1, ]
> SquidM

     Sample Year Month Location Sex    GSI
24       24    1     5        1   1 5.2970
48       48    1     5        3   1 4.2968
58       58    1     6        1   1 3.5008
60       60    1     6        1   1 3.2487
61       61    1     6        1   1 3.2304
```

< *Cut to reduce space* >

The data for females are obtained by

```
> SquidF <- Squid[Squid$Sex == 2, ]
> SquidF
  Sample Year Month Location Sex     GSI
1      1    1     1        1   2 10.4432
2      2    1     1        3   2  9.8331
3      3    1     1        1   2  9.7356
4      4    1     1        1   2  9.3107
5      5    1     1        1   2  8.9926
```

< Cut to reduce space >

The process of selecting variable data (or data frames) conditional on the values of a second variable is called conditional selection. The `unique` command applied on `Squid$Location` shows that there are four locations coded as 1, 2, 3, and 4. To extract the data from location 1, 2, or 3 we can use the following statements that all give the same result (the | symbol stands for the Boolean "or" and the != for "not equal").

```
> Squid123 <- Squid[Squid$Location == 1 |
                    Squid$Location == 2 | Squid$Location == 3, ]
> Squid123 <- Squid[Squid$Location != 4, ]
> Squid123 <- Squid[Squid$Location < 4, ]
> Squid123 <- Squid[Squid$Location <= 3, ]
> Squid123 <- Squid[Squid$Location >= 1 &
                     Squid$Location <= 3, ]
```

You can choose any of these options. Next we use the "&," which is the Boolean "and" operator. Suppose we want to extract the male data from location 1. This means that the data have to be both from male squid *and* from location 1. The following code extracts data that comply with these conditions.

```
> SquidM.1 <- Squid[Squid$Sex == 1 &
                    Squid$Location == 1,]
     Sample Year Month Location Sex    GSI
24       24    1     5        1   1 5.2970
58       58    1     6        1   1 3.5008
60       60    1     6        1   1 3.2487
61       61    1     6        1   1 3.2304
63       63    1     6        1   1 3.1848
```

< Cut to reduce space >

The data from males *and* from location 1 *or* 2 are given by

```
> SquidM.12 <- Squid[Squid$Sex == 1 &
        (Squid$Location == 1 | Squid$Location == 2), ]
```

Do *not* use the following command.

```
> SquidM <- Squid[Squid$Sex == 1, ]
> SquidM1 <- SquidM[Squid$Location == 1, ]
> SquidM1
   Sample Year Month Location Sex    GSI
24     24    1     5        1   1 5.2970
58     58    1     6        1   1 3.5008
```

```
60      60     1      6         1    1 3.2487
61      61     1      6         1    1 3.2304
62      62     1      5         3    1 3.2263
...
NA.1113      NA    NA     NA         NA   NA      NA
NA.1114      NA    NA     NA         NA   NA    NA
NA.1115      NA    NA     NA         NA   NA    NA
NA.1116      NA    NA     NA         NA   NA    NA
```

The first line extracts the male data and allocates it to `SquidM`, which is therefore of a smaller dimension (fewer rows) than `Squid` (assuming there are female squid in the data). On the next line, the Boolean vector `Squid$Location == 1` is longer than the number of rows in `SquidM`, and R will add extra rows with NAs to `SquidM`. As a result, we get a data frame, `SquidM1`, that contains NAs. The problem is that we are trying to access elements of `SquidM` using a Boolean vector that has the same number of rows as `Squid`.

Don't panic if the output of a subselecting command shows the following message.

```
> Squid[Squid$Location == 1 & Squid$Year == 4 &
        Squid$Month == 1, ]
[1] Sample    Year      Month     Location  Sex
       GSI       fSex      fLocation
<0 rows> (or 0-length row.names)
```

This simply means that no measurements were taken at location 1 in month 1 of the fourth year.

3.2.1 Sorting the Data

In addition to extracting subsets of data, at times it is useful to rearrange the data. For the squid data, you may want to sort the GSI data from low to high values of the variable "month", even if only for a quick observation. The following code can be used.

```
> Ord1 <- order(Squid$Month)
> Squid[Ord1, ]
 Sample Year Month Location Sex     GSI
1      1    1     1        1   2 10.4432
2      2    1     1        3   2  9.8331
3      3    1     1        1   2  9.7356
4      4    1     1        1   2  9.3107
5      5    1     1        1   2  8.9926
```

< Cut to reduce space >

As we are manipulating the rows of `Squid`, we need to put `Ord1` *before* the comma. We can also perform this exercise on only one variable, for instance the GSI. In this case, use

```
> Squid$GSI[Ord1]
 [1]  10.4432  9.8331  9.7356  9.3107  8.9926  8.7707
 [7]   8.2576  7.4045  7.2156  6.3882  6.0726  5.7757
[13]   1.2610  1.1997  0.8373  0.6716  0.5758  0.5518
[19]   0.4921  0.4808  0.3828  0.3289  0.2758  0.2506
```

< Cut to reduce space >

Do Exercise 2 in Section 3.7. This is an exercise in using the `read.table` function and accessing subsets from a data frame using a deep sea research dataset.

3.3 Combining Two Datasets with a Common Identifier

So far, we have seen examples in which all data points were stored in the same file. However, this may not always be the case. The authors of this book have been involved in various projects in which the data consisted of different types of measurements on the same animals. For example, in one project the measurements were made on approximately 1000 fish at different research institutes; one institute calculated morphometric measurements, another measured chemical variables, and yet another counted number of parasites. Each institute created its own spreadsheet containing the workgroup-specific variables. The crucial point was that, at each institute, researchers measured each fish, so all spreadsheets contained a column identifying the fish. Some fish were lost during the process or were unsuitable for certain procedures. Hence, the end result was a series of Excel spreadsheets, each with thousands of observations on 5–20 group-specific variables, but with a common identifier for the individual fish (case).

As a simple example of such a dataset, see the spreadsheets in Fig. 3.2. Imagine that the squid data were organised in this way, two different files or worksheets but with a common identifier. The task is now to merge the two datasets in such a way that data for sample *j* in the first dataset are put next to the data for sample *j* in the second dataset. For illustrative purposes, we have removed the fourth row from the second spreadsheet; just assume that someone forgot to type in Year, Month, Location, and Sex for observation 4. R has a useful tool for merging files, namely the *merge* function. It is run using the following code. The first two lines are used to read the two separate squid files:

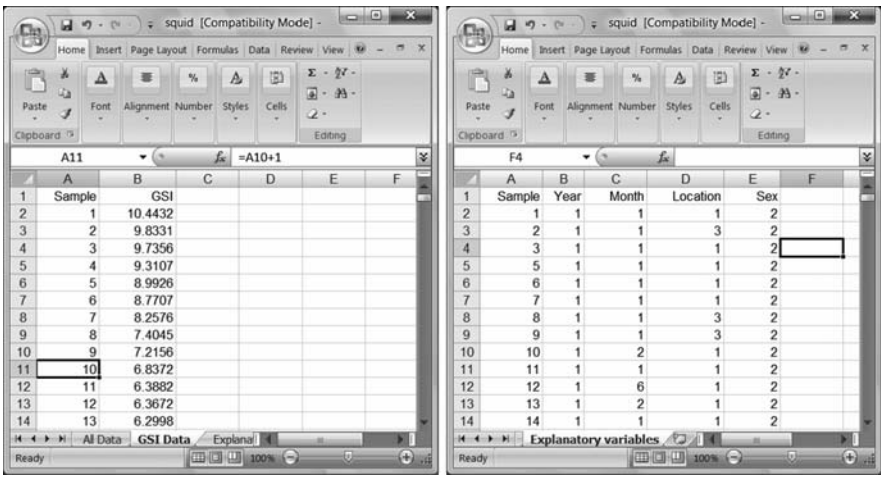

Fig. 3.2 GSI data with sample number (*left*) and the other variables with sample number (*right*). To illustrate the `merge` function, we deleted row number four in the right-hand spreadsheet

```
> setwd("C:/RBook/")
> Sq1 <- read.table(file = "squid1.txt",
                    header = TRUE)
> Sq2 <- read.table(file = "squid2.txt",
                    header = TRUE)
> SquidMerged <- merge(Sq1, Sq2, by = "Sample")
> SquidMerged
 Sample      GSI Year Month Location Sex
1      1 10.4432    1     1        1   2
2      2  9.8331    1     1        3   2
3      3  9.7356    1     1        1   2
4      5  8.9926    1     1        1   2
5      6  8.7707    1     1        1   2
6      7  8.2576    1     1        1   2
7      8  7.4045    1     1        3   2
8      9  7.2156    1     1        3   2
9     10  6.8372    1     2        1   2
10    11  6.3882    1     1        1   2
```

< Cut to reduce space >

The `merge` command takes as argument the two data frames `Sq1` and `Sq2` and combines the two datasets using as a common identifier the variable `Sample`. A useful option within the merge function is `all`. By default it is set to FALSE, which means that rows in which either `Sq1` or `Sq2` has missing

values are omitted. When set to TRUE, NAs are filled in if Sq1 has no data for a sample that is present in Sq2, and vice versa. Using this option, we get

```
> SquidMerged <- merge(Sq1, Sq2, by = "Sample",
                       all = TRUE)
> SquidMerged
 Sample     GSI Year Month Location Sex
1      1 10.4432    1     1        1   2
2      2  9.8331    1     1        3   2
3      3  9.7356    1     1        1   2
4      4  9.3107   NA    NA       NA  NA
5      5  8.9926    1     1        1   2
6      6  8.7707    1     1        1   2
7      7  8.2576    1     1        1   2
8      8  7.4045    1     1        3   2
9      9  7.2156    1     1        3   2
10    10  6.8372    1     2        1   2
```

< Cut to reduce space >

Note the missing values for Year, Month, Location, and Sex for observation (fish/case) 4. To avoid confusion, recall that we only removed the observations from row four for illustrative purposes. Further options and examples are given in the merge help file.

3.4 Exporting Data

In addition to the read.table command, R also has a write.table command. With this function, you can export numerical information to an ascii file. Suppose you extracted the male squid data, and you want to export it to another software package, or send it to a colleague. The easiest way to do this is to export the male squid data to an ascii file, then import it to the other software package, or email it to your colleague. The following commands extract the male data (in case you didn't type it in yet), and exports the data to the file, MaleSquid.txt.

```
> SquidM <- Squid[Squid$Sex == 1, ]
> write.table(SquidM,
    file = "MaleSquid.txt",
    sep = " ", quote = FALSE, append = FALSE, na = "NA")
```

The first argument in the write.table function is the variable that you want to export, and, obviously, you also need a file name. The sep = " " ensures that the data are separated by a space, the quote = FALSE avoids

quotation marks around character strings (headings), `na = "NA"` allows you to specify how missing values are represented, and `append = FALSE` opens a new file. If it were set to `TRUE`, it would append the variable `SquidM` to the end of an existing file.

Let us illustrate some of these options. When we run the code above, the first six lines in the ascii file `MaleSquid.txt` are as follows.

```
Sample Year Month Location Sex GSI fLocation fSex
24 24 1 5 1 1 5.297 1 M
48 48 1 5 3 1 4.2968 3 M
58 58 1 6 1 1 3.5008 1 M
60 60 1 6 1 1 3.2487 1 M
61 61 1 6 1 1 3.2304 1 M
```

< Cut to reduce space >

Hence, the elements are separated by a space. Note that we are missing the name of the first column. If you import these data into Excel, you may have to shift the first row one column to the right. We can change the `sep` and `quote` options:

```
> write.table(SquidM,
   file = "MaleSquid.txt",
   sep = ",", quote = TRUE, append = FALSE, na = "NA")
```

It gives the following output in the ascii file `MalesSquid.txt`.

```
"Sample","Year","Month","Location","Sex","GSI",
"fLocation","fSex"
"24",24,1,5,1,1,5.297,"1","M"
"48",48,1,5,3,1,4.2968,"3","M"
"58",58,1,6,1,1,3.5008,"1","M"
"60",60,1,6,1,1,3.2487,"1","M"
"61",61,1,6,1,1,3.2304,"1","M"
```

The fact that the headers extend over two lines is due to our text editor. The real differences are the comma separations and the quotation marks around categorical variables, and headers and labels. For some packages this is important. The `append = TRUE` option is useful if, for example, you have to apply linear regression on thousands of datasets and you would like to have all the numerical output in one file.

Do Exercise 3 in Section 3.7. This is an exercise in the `write.table` function using a deep sea research dataset.

3.5 Recoding Categorical Variables

In Section 3.1, we used the `str` function to give the following output for the Squid data frame.

```
> str(Squid)
'data.frame': 2644 obs. of 6 variables:
 $ Sample   : int 1 2 3 4 5 6 7 8 9 10 ...
 $ Year     : int 1 1 1 1 1 1 1 1 1 1 ...
 $ Month    : int 1 1 1 1 1 1 1 1 1 2 ...
 $ Location: int 1 3 1 1 1 1 1 3 3 1 ...
 $ Sex      : int 2 2 2 2 2 2 2 2 2 2 ...
 $ GSI      : num 10.44 9.83 9.74 9.31 8.99 ...
```

The variable `Location` is coded as 1, 2, 3, or 4, and `Sex` as 1 or 2. Such variables are categorical or nominal variables. In Excel, we could have coded sex as male and female. It is good programming practice to create new variables in the data frame that are recoded as nominal variables, for example:

```
> Squid$fLocation <- factor(Squid$Location)
> Squid$fSex <- factor(Squid$Sex)
```

These two commands create two new variables inside the data frame `Squid`, `fLocation` and `fSex`. We used the f in front of the variable name to remind us that these are nominal variables. In R, we can also call them factors, hence the f. Type

```
> Squid$fSex
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[18] 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2
[35] 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2
 ...
[2602] 1 2 1 1 2 1 1 1 2 1 2 1 1 2 1 1 2
[2619] 1 2 2 1 1 1 1 1 1 1 1 1 2 1 1 1 2
[2636] 1 2 1 2 1 2 1 1 1
Levels: 1 2
```

Note the extra line at the end. It tells us that `fSex` has two levels, 1 and 2. It is also possible to relabel these levels as "male" and "female", or, perhaps more efficiently, M and F:

```
> Squid$fSex <- factor(Squid$Sex, levels = c(1, 2),
  labels = c("M", "F"))
> Squid$fSex
 [1] F F F F F F F F F F F F F F F F F F
[18] F F F F F F M F F F F F F F F F F F
[35] F F F F F F F F F F F F F M F F F
 ...
```

```
[2602] M F M M F M M M F M F M M F M M F
[2619] M F F M M M M M M M M F M M M F
[2636] M F M F M F M M M
Levels: M F
```

Every 1 has been converted to an "M", and every 2 to an "F". You can now use `fSex` in functions such as `lm` or `boxplot`:

```
> boxplot(GSI ~ fSex, data = Squid) #Result not shown
> M1 <- lm(GSI ~ fSex + fLocation, data = Squid)
```

Another advantage of using predefined nominal variables in a linear regression function is that its output becomes much shorter. Although we do not show the output here, compare that of the following commands.

```
> summary(M1)
> M2 <- lm(GSI ~ factor(Sex) + factor(Location),
     data = Squid)
> summary(M2)
```

The estimated parameters are identical, but the second model needs more space on the screen (and on paper). This becomes a serious problem with second- and third-order interaction terms.

Instead of the command `factor`, you can also use `as.factor`. To convert a factor to a numerical vector, use `as.numeric`. This can be useful for making plots with different colours for males and females (if you have lost, for some reason, the original vector, Sex). See also Chapter 5.

The same can be done for `fLocation`:

```
> Squid$fLocation
 [1] 1 3 1 1 1 1 1 3 3 1 1 1 1 1 1 1 3
[18] 1 3 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1
[35] 1 1 1 1 1 3 1 1 1 1 3 1 1 3 1 1 1
 ...
[2602] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[2619] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[2636] 1 1 1 1 1 1 1 1 1
Levels: 1 2 3 4
```

Note that this nominal variable has four levels. In this case, the levels are sorted from small to large. And this means that in a boxplot, the data from location 1 are next to location 2, 2 is next to 3, and so on. Sometimes it can be useful to change the order (e.g., in an `xyplot` function from the `lattice` package). This is done as follows.

```
> Squid$fLocation <- factor(Squid$Location,
                            levels = c(2, 3, 1, 4))
```

```
> Squid$fLocation
 [1] 1 3 1 1 1 1 1 3 3 1 1 1 1 1 1 1 3
[18] 1 3 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1
[35] 1 1 1 1 1 3 1 1 1 1 3 1 1 3 1 1 1
...
[2602] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[2619] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[2636] 1 1 1 1 1 1 1 1 1
Levels: 2 3 1 4
```

The data values remain the same, but a command such as

```
> boxplot(GSI ~ fLocation, data = Squid)
```

produces a slightly different boxplot because the order of the levels is different. Relevelling is also useful for conducting a posthoc test in linear regression (Chapter 10 in Dalgaard, 2002).

We began this chapter with selecting the male data:

```
> SquidM <- Squid[Squid$Sex == 1, ]
```

We can also do this with `fSex`, but now we need:

```
> SquidM <- Squid[Squid$fSex == "1", ]
```

The quotation marks around the 1 are needed because `fSex` is a factor. The effect of defining new nominal variables can also be seen with the `str` command:

```
> Squid$fSex <- factor(Squid$Sex, labels = c("M", "F"))
> Squid$fLocation <- factor(Squid$Location)
> str(Squid)
'data.frame': 2644 obs. of 8 variables:
 $ Sample   : int 1 2 3 4 5 6 7 8 9 10 ...
 $ Year     : int 1 1 1 1 1 1 1 1 1 1 ...
 $ Month    : int 1 1 1 1 1 1 1 1 1 2 ...
 $ Location : int 1 3 1 1 1 1 1 3 3 1 ...
 $ Sex      : int 2 2 2 2 2 2 2 2 2 2 ...
 $ GSI      : num 10.44 9.83 9.74 9.31 8.99 ...
 $ fSex     : Factor w/ 2 levels "M","F": 2 2 2 2 2 ...
 $ fLocation: Factor w/ 4 levels "1","2","3","4": 1 ...
```

Note that `fSex` and `fLocation` are now factors (categorical variables), and the levels are shown. Any function will now recognise them as factors, and there is no further need to use the `factor` command with these two variables.

Do Exercise 4 in Section 3.7. This is an exercise in use of the `factor` function using a deep sea research dataset.

3.6 Which R Functions Did We Learn?

Table 3.1 shows the R commands introduced in this chapter.

Table 3.1 R functions introduced in this chapter

Function	Purpose	Example
`write.table`	Write a variable to an ascii file	`write.table(Z,file=''test.txt'')`
`order`	Determine the order of the data	`order(x)`
`merge`	Merge two data frames	`merge(x,y,by=''ID'')`
`attach`	Make variables inside a data frame available	`attach(MyData)`
`str`	Shows the internal structure of an object	`str(MyData)`
`factor`	Defines a variable as a factor	`factor(x)`

3.7 Exercises

Exercise 1. Using the `read.table` function and accessing variables from a data frame with epidemiological data.

The file *BirdFlu.xls* contains the annual number of confirmed cases of human Avian Influenza A/(H5N1) for several countries reported to the World Health Organization (WHO). The data were taken from the WHO website (www.who.int/en/) and reproduced for educational purposes. Prepare the spreadsheet and import these data into R. If you are a non-Windows user, start with the file *BirdFlu.txt*. Note that you will need to adjust the column names and some of the country names.

Use the `names` and `str` command in R to view the data. Print the number of bird flu cases in 2003. What is the total number of bird flu cases in 2003 and in 2005? Which country has had the most cases? Which country has had the least bird flu deaths?

Using methods from Chapter 2, what is the total number of bird flu cases per country? What is the total number of cases per year?

Exercise 2. Using the `read.table` function and accessing subsets of a data frame with deep sea research data.

If you have not completed Exercise 6 in Chapter 2, do so and import the data from the *ISIT.xls* file.

In R, extract the data from station 1. How many observations were made at this station? What are the minimum, median, mean, and maximum sampled depth at station 1? What are the minimum, median, mean, and maximum sampled depth at station 2? At station 3?

Identify any stations with considerably fewer observations. Create a new data frame omitting these stations.

Extract the data from 2002. Extract the data from April (of all years). Extract the data that were measured at depths greater than 2000 meters (from all years and months). Show the data according to increasing depth values.

Show the data that were measured at depths greater than 2000 meters in April.

Exercise 3. Using the `write.table` function with deep sea research data.

In the final step of the previous exercise, data measured at depths greater than 2000 meters in April were extracted. Export these data to a new ascii file.

Exercise 4. Using the `factor` function and accessing subsets of a data frame with deep sea research data.

Stations 1 through 5 were sampled in April 2001, stations 6 through 11 in August 2001, stations 12 through 15 in March 2002, and stations 16 through 19 in October 2002. Create two new variables in R to identify the month and the year. Note that these are factors. Do this by adding the new variables inside the data frame.