Chapter 9
Common R Mistakes

The following addresses avoiding some errors that we see on a regular basis during our R courses.

9.1 Problems Importing Data

9.1.1 Errors in the Source File

The code required to import data into R was discussed in Chapter 2. The first major task is ensuring that the spreadsheet (or ascii file) is adequately prepared. Do not use spaces in variable names or include blank cells. The error messages that will result were shown in Chapter 2, and are not repeated here.

If your column names are species names of the form *Delphinus delphi*, call it *Delphinus.delphi* with a point between the two names, *Delphinus_delphi* (underscore), or, better yet, something shorter, such as *Ddelphi*.

9.1.2 Decimal Point or Comma Separation

Another potential pitfall is the decimal separation used: comma or point. We often teach groups in which some participants have computers with point separation and the rest use commas. In Chapter 2, we demonstrated use of the `dec` option in the `read.table` function to set the style of separation. *Always* use the `str` function after importing the data to verify that the data have been imported as intended. If you import the data using the incorrect `dec` option, R will accept it without an error message. The difficulties arise later when you attempt to work with the data, for example, to make a boxplot or take the mean of a variable which is continuous but has erroneously been imported as a categorical variable because of the wrong `dec` option.

The problems may be compounded by the fact that the mistake is not always readily apparent, because you may occasionally get away with using the wrong decimal separator. In the following example, the first two commands import the

A.F. Zuur et al., *A Beginner's Guide to R*, Use R,
DOI 10.1007/978-0-387-93837-0_9, © Springer Science+Business Media, LLC 2009

cod parasite data that were used in Chapter 6. Note that we used the `dec = ","` option in the `read.table` command, although the ascii file contains data with decimal point separation.

```
> setwd("c:/RBook/")
> Parasite <- read.table(file = "CodParasite.txt",
                  header = TRUE, dec = ",")
```

The `str` function shows the imported data:

```
> str(Parasite)
'data.frame'  : 1254 obs. of 11 variables:
$Sample     : int 1 2 3 4 5 6 7 8 9 10 ...
$Intensity  : int 0 0 0 0 0 0 0 0 0 0 ...
$Prevalence : int 0 0 0 0 0 0 0 0 0 0 ...
$Year       : int 1999 1999 1999 1999 1999 ...
$Depth      : int 220 220 220 220 220 220 220 ...
$Weight     : Factor w/ 912 levels "100",..: 159...
$Length     : int 26 26 27 26 17 20 19 77 67 ...
$Sex        : int 0 0 0 0 0 0 0 0 0 0 ...
$Stage      : int 0 0 0 0 0 0 0 0 0 0 ...
$Age        : int 0 0 0 0 0 0 0 0 0 0 ...
```

`Length` has been correctly imported as an integer, but the variable `Weight` is considered a categorical variable. This is because some of the weight values are written with decimals (e.g., 148.0), whereas all other variables are coded as integers in the text file. This means that the following commands will work.

```
> mean(Parasite$Intensity, na.rm = TRUE)
[1] 6.182957
> boxplot(Parasite$Intensity) #Result not shown here
```

However, entering the same code for `Weight` gives error messages:

```
> mean(Parasite$Weight)
[1] NA
Warning message:
In mean.default(Parasite$Weight): argument is not numeric
or logical: returning NA

> boxplot(Parasite$Weight)

Error in oldClass(stats) <- cl: adding class "factor" to
an invalid object
```

If you use `Weight` as a covariate in linear regression, you may be surprised at the large number of regression parameters that it consumes; `Weight` was automatically fitted as a categorical variable. It is only by chance that the mean and boxplot functions using `Intensity` were accurately produced; if it had contained values including decimals, the same error message would have appeared.

9.1.3 Directory Names

Problems may also arise when importing data with directory names containing non-English alphabetical characters such as á, , , and many more. This is a language issue that may be difficult to resolve if you are working with datasets contributed by colleagues using other alphabet systems. Surprisingly, problems do not occur on all computers. It is advisable to keep the directory structure simple and avoid characters in the file and directory names that your computer may see as "strange".

9.2 Attach Misery

When conducting a course, we face the dilemma of whether to teach a quick-and-easy approach to accessing variables in a data frame using the `attach` function, to put participants through some extra R coding using the `data` argument (when applicable), or to teach the use of the `$` notation. This is a slightly controversial area, as some authorities insist that the `attach` function should absolutely never be used, whereas others have written books in which the function is used extensively (e.g., Wood, 2006). When we work with a single dataset, we use the `attach` command, as it is more convenient. However, there are rules that must be followed, and we see many R novices breaking these rules.

9.2.1 Entering the Same `attach` Command Twice

The most common problem incurred with the `attach` function arises when one program's code containing the `attach` command runs it in R, detects a programming mistake, fixes it, and proceeds to rerun the entire piece of code. Here is an example:

```
> setwd("c:/RBook/")
> Parasite <- read.table(file = "CodParasite.txt",
                         header = TRUE)
> attach(Parasite)
> Prrrarassite
Error: object "Prrrarassite" not found
```

Because we misspelled *Parasite*, R gives an error message. The obvious response is to correct the typing error and try again. However, if we correct

the mistake in the text editor (e.g., Tinn-R) and then resend, or copy, all the code (which includes the attach command) to R, it will result in the following.

```
> setwd("c:/RBook/")
> Parasite <- read.table(file = "CodParasite.txt",
                         header = TRUE)
> attach(Parasite)
The following object(s) are masked from Parasite (posi-
tion 3):
Age Area Depth Intensity Length Prevalence Sample Sex
Stage Weight Year
```

At this point it is useful to consult the help file of the attach function. It says that the function adds the data frame Parasite to its search path, and, as a result, the variables in the data frame can be accessed without using the $ notation. However, by attaching the data frame twice, we have made available two copies of each variable. If we make changes to, for example, Length and, subsequently, use Length in a linear regression analysis, we have no way of ensuring that the correct value is used.

The alternative is to use the detach function before rerunning attach (the code below assumes that we have not yet used the attach function):

```
> setwd("c:/RBook/")
> Parasite <- read.table(file = "CodParasite.txt",
                         header = TRUE)
> attach(Parasite)
```

We can now begin programming; to detach the data frame Parasite, use:

```
> detach(Parasite)
```

Another procedure to avoid is running a for loop that in each iteration attaches a data frame, as this will generate an expanding search path which will eventually slow down your computer. The help file for the with function also provides an alternative to the attach function.

9.2.2 Attaching Two Data Frames Containing the Same Variable Names

Suppose we import the cod parasite data and the squid data and employ the attach function to make variables in both data frames available, using the following code.

```
> setwd("c:/RBook/")
> Parasite <- read.table(file = "CodParasite.txt",
                         header = TRUE)
> Squid <- read.table(file = "Squid.txt", header=TRUE)
> names(Parasite)
[1] "Sample"  "Intensity"   "Prevalence"    "Year"
[5] "Depth"   "Weight"      "Length"        "Sex"
[9] "Stage"   "Age"         "Area"

> names(Squid)

[1] "Sample" "Year" "Month" "Location" "Sex"
[6] "GSI"

> attach(Parasite)
> attach(Squid)

The following object(s) are masked from Parasite:
Sample Sex Year

> boxplot(Intensity ~ Sex)

Error in model.frame.default(formula=Intensity ~ Sex):
variable lengths differ (found for 'Sex')

> lm(Intensity ~ Sex)

Error in model.frame.default(formula = Intensity ~ Sex,
drop.unused.levels = TRUE): variable lengths differ
(found for 'Sex')
```

The first three commands import the data. The output of the two `names` functions show that both data frames contain the variable `Sex`. We used the `attach` function to make the variables in both data frames available. To see the effect of this, we can make a boxplot of the `Intensity` data conditional on `Sex`. The error message generated by the `boxplot` function shows that the length of the vectors `Intensity` and `Sex` differ. This is because R has used `Intensity` from the `Parasite` data frame and `Sex` from the `Squid` data frame. Imagine what would have happened if, fortuitously, these two variables were of the same dimension: we would have modelled the number of parasites in cod measured in the Barents Sea as an effect of sex of squid from the North Sea!

9.2.3 Attaching a Data Frame and Demo Data

Many statistics textbooks come with a package that contains datasets used in the book, for example, the `MASS` package from Venables and Ripley (2002), the

nlme package from Pinheiro and Bates (2000), and the mgcv package from Wood (2006), among many others. The typical use of such a package is that the reader accesses the help file of certain functions, goes to the examples at the end of a help file, and copies and runs the code to see what it does. Most often, the code from a help file loads a dataset from the package using the data function, or it creates variables using a random number generator. We have also seen help file examples that contain the attach and detach functions. When using these it is not uncommon to neglect to copy the entire code; the detach command may be omitted, leaving the attach function active. Once you understand the use of the demonstrated function, it is time to try it with your own data. If you have also applied the attach function to your data, you may end up in the scenario outlined in the previous section.

Problems may also occur if demonstration data loaded with the data function contain some of the same variable names used as your own data files.

The general message is to be careful with the attach function, and use clear and unique variable names.

9.2.4 *Making Changes to a Data Frame After Applying the* `attach` Function

The following example is something that we see on a regular basis. Often, our course participants own multiple R books, which may recommend different R styles. For example, one book may use the attach function, whereas another uses a more sophisticated method of accessing variables from a data frame. Mixing programming styles can sometimes cause trouble, as can be seen from the example below.

```
> setwd("c:/RBook/")
> Parasite <- read.table(file = "CodParasite.txt",
                         header = TRUE)
> Parasite$fSex <- factor(Parasite$Sex)
> Parasite$fSex
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[21] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 1 1
  ...
> attach(Parasite)
> fSex
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[21] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 1 1
  ...
> Parasite$fArea <- factor(Parasite$Area)
> fArea
Error: object "fArea" not found
```

On the first three lines, the data are imported and a new categorical variable fSex is created inside the data frame Parasite. We then make all variables in this data frame available with the attach function, and, consequently, we can access the fSex variable by simply typing it in. The numerical output shows that this was successful. If we subsequently decide to convert Area into a new categorical variable, fArea inside the data frame Parasite, we encounter a problem. We cannot access this variable by typing its name into the console (see the error message). This is because the attach function has been executed, and variables added to Parasite afterwards are not available. Possible solutions are:

1. Detach the data frame Parasite, add fArea to the data frame Parasite, and attach it again.
2. Define fArea before using the attach function.
3. Define fArea outside the data frame.

9.3 Non-attach Misery

In addition to the attach function, there are various other options available for accessing the variables in a data frame. We discussed the use of the data argument and the $ symbol in Chapter 2. In the latter case, we can use

```
> setwd("c:/RBook/")
> Parasite <- read.table(file = "CodParasite.txt",
                         header = TRUE)
> M0 <- lm(Parasite$Intensity ~
             Parasite$Length * factor(Parasite$Sex))
```

The first two lines import the cod parasite data. The last two lines apply a linear regression model in which Intensity is modelled as a function of length and sex of the host. We do not discuss linear regression nor its output here. It is sufficient to know that the function has the desired effect; type summary(M0) to see the output. Note that we used the Parasite $ notation to access variables in the data frame Parasite (see Chapter 2). The following two commands load the nlme package and apply linear regression using the generalised least squares function gls (Pinheiro and Bates, 2002).

```
> library(nlme)
> M1 <- gls(Parasite$Intensity ~
                Parasite$Length * factor(Parasite$Sex))
Error in eval(expr, envir, enclos): object "Intensity"
not found
```

The results obtained by the lm and gls functions should be identical, yet R (regardless of the version used) gives an error message for the latter.

The solution is to use the `data` argument and avoid using the `Parasite$` notation in the `gls` function.

9.4 The Log of Zero

The following code looks correct. It imports the dataset of the cod parasite data and applies a logarithmic transformation on the number of parasites in the variable `Intensity`.

```
> setwd("c:/RBook/")
> Parasite <- read.table(file = "CodParasite.txt",
                         header = TRUE)
> Parasite$LIntensity <- log(Parasite$Intensity)
```

There is no error message, but, if we make a boxplot of the log-transformed values, problems become apparent; see the left boxplot in Fig. 9.1. The difficulty arises because some fish have zero parasites, and the log of zero is not defined, as can be seen from inspecting the values:

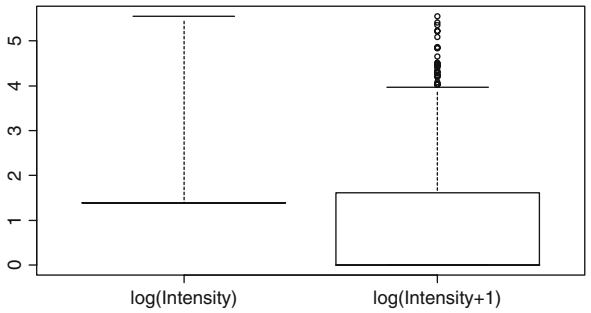

Fig. 9.1 Boxplot of the log-transformed `Intensity` (*left*) and the log-transformed Intensity after adding a constant value of 1 to avoid problems with the log of zero (*right*)

```
> Parasite$LIntensity
 [1]    -Inf    -Inf    -Inf    -Inf    -Inf
 [6]    -Inf    -Inf    -Inf    -Inf    -Inf
[11]    -Inf    -Inf    -Inf    -Inf    -Inf
...
[1246] 4.0073332 4.3174881 4.4308168 4.4886364
[1251] 4.6443909 4.8283137 4.8520303 5.5490761
```

Carrying out linear regression with the variable `LIntensity` results in a rather intimidating error message:

```
> M0 <- lm(LIntensity ~ Length * factor(Sex),
           data = Parasite)
```

```
Error in lm.fit(x, y, offset  =  offset, singular.ok =
singular.ok,...): NA/NaN/Inf  in foreign function call
(arg 4)
```

The solution is to add a small constant value to the `Intensity` data, for example, 1. Note that there is an on-going discussion in the statistical community concerning adding a small value. Be that as it may, you cannot use the log of zero when doing calculations in R. The following code adds the constant and draws the boxplot shown on the right side in Fig. 9.1.

```
> Parasite$L1Intensity <- log(Parasite$Intensity + 1)
> boxplot(Parasite$LIntensity, Parasite$L1Intensity,
       names = c("log(Intensity)", "log(Intensity+1)"))
```

To reiterate, you should not take the log of zero!

9.5 Miscellaneous Errors

In this section, we present some trivial errors that we see on a regular basis.

9.5.1 The Difference Between 1 and l

Look at the following code. Can you see any differences between the two plot functions? The first one is valid and produces a simple graph; the second plot function gives an error message.

```
> x <- seq(1, 10)
> plot(x, type = "l")
> plot(x, type = "1")
Error in plot.xy(xy, type, ...) : invalid plot type '1'
```

The text in the section title may help to answer the question, as its font shows more clearly the difference between the 1 (one) and the l ("ell"). In the first function, the `l` in `type = "l"` stands for line, whereas, in the second plot function, the character in `type = "1"` is the numeral `1` (this is an R syntax error). If this text is projected on a screen in a classroom setting, it is difficult to detect any differences between the l and 1.

9.5.2 The Colour of 0

Suppose you want to make a Cleveland dotplot of the variable `Depth` in the cod parasite data to see the variation in depths from which fish were

sampled (Fig. 9.2A). All fish were taken from depths of 50–300 meters. In addition to the numbers of parasites, we also have a variable, Prevalence, which indicates the presence (1) or absence (0) of parasites in a fish. It is interesting to add this information to the Cleveland dotplot, for example, by using different colours to denote Prevalence. This is shown in panel B. The code we use is as follows (assuming the data to have been imported as described in previous sections).

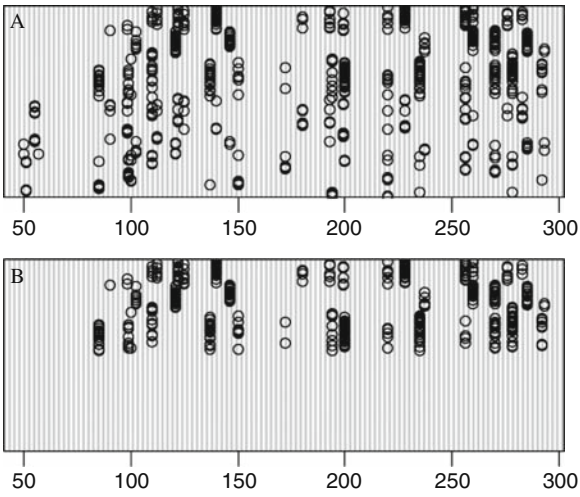

Fig. 9.2 **A**: Cleveland dotplot of depth. The horizontal axis shows depth, and the vertical axis represents the order of the observations as imported from the text file. **B**: Same as panel A, with points coloured based on the values of `Prevalence`

```
> par(mfrow = c(2, 1), mar = c(3, 3, 2, 1))
> dotchart(Parasite$Depth)
> dotchart(Parasite$Depth, col = Parasite$Prevalence)
```

We encounter a problem, in that some of the points have disappeared. This is because we used a variable in the `col` option that has values equal to 0, which would represent a lack of colour. It is better to use something along the lines of `col = Parasite$Prevalence + 1`, or define a new variable using appropriate colours.

9.5.3 Mistakenly Saved the R Workspace

Last, but not least, we deal with problems arising from mistakenly saving the workspace. Suppose that you loaded the owl data that was used in Chapter 7:

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

To see which variables are available in the workspace, type:

```
> ls()
[1] "Owls"
```

The `ls` command gives a list of all objects (after an extended work session, you may have a lot of objects).

You now decide to quit R and click on **File -> Exit**. The window in Fig. 9.3 appears. We always advise choosing "No," not saving, instead rerunning the script code from the text editor (e.g., Tinn-R) when you wish to work with it again. The only reason for saving the workspace is when running the calculations is excessively time consuming. It is easy to end up with a large number of saved workspaces, the contents of which are complete mysteries. In contrast, script code can be documented.

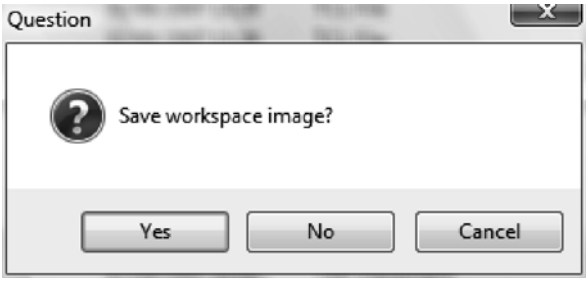

Fig. 9.3 Window asking the user whether the workspace should be saved before closing R

However, suppose that you do click on "Yes." Dealing with this is easy. The directory C:/RBook will contain a file with the extension .RData. Open Windows Explorer, browse to the working directory (in this case: C:/RBook) and delete the file with the big blue R.

Things are more problematical if, instead of using the `setwd` command, you have entered:

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

If you now quit, saving the workspace, when R is started again the following text will appear.

```
R version 2.7.2 (2008-08-25)
Copyright (C) 2008 The R Foundation for Statistical
Computing

ISBN 3-900051-07-0
R is free software and comes with ABSOLUTELY NO WARRANTY.
```

```
You are welcome to redistribute it under certain condi-
tions.

Type 'license()' or 'licence()' for distribution
details.

  Natural language support but running in an English
locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and

'citation()' on how to cite R or R packages in publica-
tions.
Type 'demo()' for some demos, 'help()' for on-line help,
or
'help.start()' for an HTML browser interface to help.

Type 'q()' to quit R.

[Previously saved workspace restored]
>
```

It is the last line that spoils the fun. R has loaded the owl data again. To convince yourself, type:

```
> Owls
```

The owl data will be displayed. It will not only be the owl data that R has saved, but also all other objects created in the previous session. Restoring a saved workspace can cause the same difficulties as those encountered with `attach` (variables and data frames being used that you were not aware had been loaded).

To solve this problem, the easiest option is to clear the workspace (see also Chapter 1) with:

```
> rm(list = ls(all = TRUE))
```

Now quit R and save the (empty) workspace. The alternative is to locate the .RData file and manually delete it from Windows Explorer. In our computer (using VISTA), it would be located in the directory: C:/Users/UserName. Network computers and computers with XP are likely to have different settings for saving user information. The best practice is simply to avoid saving the workspace.