

Data Entry Vignette

Derek H. Ogle

May 29, 2009

Many fisheries biologists, when first learning to use R, struggle with how to “enter” their data into R. In this vignette I will show how to port your data into R from MSAccess (2007), MSAccess (pre-2007), MSEXcel (2007; .xlsx files), MSEXcel (pre-2007 .xls files), tab-delimited files, and comma-separated values (CSV) files. In addition, I will briefly mentioned how “pre-existing” data (not “your” data) can be loaded from existing R packages. Finally, I will provide a brief tutorial on how to create subsets of your data once it is in R.

This vignette is not an exhaustive treatment of how to load data into R. I have simply tried to bring together methods to handle what I see as the most common situations among fisheries students and professionals. It should be noted that the R Project does provide a [manual that describes the importing and exporting of data](#).

1 The Data Files

Throughout this vignette, I will use a dataset of typical biological data from a population of ruffe (*Gymnocephalus cernuus*). These data contain the following variables measured on 40 ruffe captured in the St. Louis River Harbor, Lake Superior in 2007:

- *fishID* : a unique fish identification number.
- *locShort* : name of the sampling location (all entries are `St. Louis R. (2007)`).
- *year* : year of capture.
- *month* : month of capture.
- *day* : day of capture.
- *date* : date of capture (this is redundant with the three previous variables but was included to illustrate how date data are read into R).
- *tl* : total length (mm) (note: has missing data).
- *wt* : weight (g).
- *sex* : sex (`female`, `male`, or `unknown`).
- *maturity* : coarse maturity stage (`immature` or `mature`) (note: has missing data).

The first few lines of these data look like:

```
fishID      locShort year month day      date  tl  wt  sex maturity
1      60 St. Louis R. (2007) 2007      9 20 9/20/2007 134 24.6 female  mature
2      61 St. Louis R. (2007) 2007      9 20 9/20/2007 111 14.7 female  mature
3      62 St. Louis R. (2007) 2007      9 20 9/20/2007 110 12.3 female  immature
4      63 St. Louis R. (2007) 2007      9 20 9/20/2007 115 16.0 female  mature
5      64 St. Louis R. (2007) 2007      9 20 9/20/2007  92  8.3 female  mature
6      65 St. Louis R. (2007) 2007      9 20 9/20/2007  88  7.8 female  mature
```

These data have been entered into a variety of formats for this vignette. These formats are as follows:

- MSAccess (2007): file called `RuffeBio.accdb`, table called `dbRuffeBio`, and query (which retrieves all records from the `dbRuffeBio` table) called `qryRuffeBio`.

- MSAccess (2002-03): file called **RuffeBio.mdb** , same table and query names as above.
- MSAccess (2000): file called **RuffeBio00.mdb** , same table and query names as above.
- MSExcel (2007): file called **RuffeBio.xlsx** , sheet called **Bio**
- MSExcel (pre-2007): file called **RuffeBio.xls** , same sheet name as above.
- Tab-delimited text: file called **RuffeBio.txt** .
- Comma-separated-values: file called **RuffeBio.csv** .

All of these files contain the same data, are available on the RForge.net FSA vignettes page, and will be used to illustrate the different methods for loading data into R in the following sections.

All functions illustrated below assume that you have changed the R working directory to the location that holds your data files. The working directory can be set through the “File..Change Dir” menu item but I prefer to set it using `setwd()` . For example, on my system, I set the working directory with

```
> setwd("c://aaaWork//web//fishR//gnrlex//DataEntry")
```

Please note that you will have to change this to the directory containing the data files on YOUR system. In addition, note the use of the two forward-slashes to separate the folders¹.

On a Windows machine you can use a dialog box to interactively select (through browsing dialog boxes) a data file by including `file.choose()` in place of any of the file names shown below. For example (this will make more sense after reading the ensuing sections), instead of

```
> txt.bio <- read.table("RuffeBio.txt", header = TRUE, sep = "\t", na.strings = "")
```

you could use

```
> txt.bio <- read.table(file.choose(), header = TRUE, sep = "\t", na.strings = "")
```

to interactively browse to where the data file is located. In this instance you would not have to worry about setting the working directory with `setwd()` .

2 Methods for Comparing Results

The following function can be used to determine if two data frames are equal or not. You do not need to be worried about the details of this function but need to know that if it returns a `TRUE` then the two data frames examined are completely equal (i.e., all comparable elements in each data frame are the same).

```
> comp.df <- function(df1, df2) {
+   pdf1 <- paste(df1)
+   pdf2 <- paste(df2)
+   all(is.element(pdf1, pdf2))
+ }
```

¹The forward-slash is used as an escape character in R and it must be “escaped” so that a forward-slash will be read.

3 Reading Directly from Microsoft Products

3.1 Access

The `RODBC` package can be used to provide an Open DataBase Connectivity (ODBC) interface through R that allows a direct port from Access or Excel to R. The functions in the `RODBC` package require application-specific drivers to be on your system. Generally, if you have the application that you are attempting to port your data from then you likely have the drivers on your system. The `RODBC` package is loaded into R with,

```
> library(RODBC)
```

A channel of communication must first be opened between R and the application which holds your data. For example, each of the following lines can be used to open an application-specific channel to the files described in Section 1. Of course, you would only use one of these commands – the one specific to where your data is stored.

```
> a07.connect <- odbcConnectAccess2007("RuffeBio.accdb") # Access 2007
> a03.connect <- odbcConnectAccess("RuffeBio.mdb")      # Access 2003
> a00.connect <- odbcConnectAccess("RuffeBio00.mdb")   # Access 2000
```

Once the connection to the data file has been created then the data must be “fetched” with `sqlFetch()`. When fetching data from an Access database `sqlFetch()` requires two arguments. The first argument is the name of the connection created above. The second argument is the name, in quotes, of the object (table or query) in Access that contains the data to be fetched. If the database is protected with a user ID and password then the user ID can be passed in the `UID=` and the password passed in the `pwd=` optional arguments. For example, the data stored in the Access (2007) file is fetched and stored into an R object with,

```
> a07d.bio <- sqlFetch(a07.connect, "dbRuffeBio")
```

The structure of the `a07d.bio` object can be seen with,

```
> str(a07d.bio)

'data.frame':      40 obs. of  10 variables:
 $ fishID  : num  60 61 62 63 64 65 66 67 68 69 ...
 $ locShort: Factor w/ 1 level "St. Louis R. (2007)": 1 1 1 1 1 1 1 1 1 1 ...
 $ year    : num  2007 2007 2007 2007 2007 ...
 $ month   : num  9 9 9 9 9 9 9 9 9 9 ...
 $ day     : num  20 20 20 20 20 20 20 20 20 20 ...
 $ date    : POSIXct, format: "2007-09-20" "2007-09-20" ...
 $ tl      : num  134 111 110 115 92 88 95 90 99 107 ...
 $ wt      : num  24.6 14.7 12.3 16 8.3 7.8 9.7 8.2 11.7 13 ...
 $ sex     : Factor w/ 3 levels "female","male",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ maturity: Factor w/ 3 levels "", "immature",,..: 3 3 2 3 3 3 3 3 3 3 ...
```

Note that one of the levels of `maturity` is the “empty string”,

```
> levels(a07d.bio$maturity)

[1] ""          "immature" "mature"
```

The “empty string” appears as a level of *maturity* because the stage of maturity was not recorded for every fish in the database. The empty string can be considered as missing (NA in the R lexicon) by including the `na.string=""` argument into `sqlFetch()`. Thus,

```
> a07d.bio2 <- sqlFetch(a07.connect, "dbRuffeBio", na.string = "")
> levels(a07d.bio2$maturity)

[1] "immature" "mature"
```

It should be noted that missing fields in numeric variables are automatically converted to NA (e.g., see *tl* in row 39 below),

```
> tail(a07d.bio2)

  fishID      locShort year month day      date  tl  wt  sex maturity
35    94 St. Louis R. (2007) 2007     9  20 2007-09-20  81  5.8 female  mature
36    95 St. Louis R. (2007) 2007     9  20 2007-09-20  81  5.8  male  mature
37    96 St. Louis R. (2007) 2007     9  20 2007-09-20  65  2.9 female  mature
38    97 St. Louis R. (2007) 2007     9  20 2007-09-20  42  0.8 unknown immature
39    98 St. Louis R. (2007) 2007     9  20 2007-09-20  NA 10.5  male  mature
40    99 St. Louis R. (2007) 2007     9  20 2007-09-20 115 17.9 female  mature
```

The data can be read from the query in the Access (2007) file in a similar manner,

```
> a07q.bio <- sqlFetch(a07.connect, "qryRuffeBio", na.string = "")
```

which results in an exactly equivalent data frame,

```
> comp.df(a07d.bio2, a07q.bio)

[1] TRUE
```

The queries (as an example, tables could be fetched similarly) from the Access (2003) and Access (2000) files are fetched and displayed similarly,

```
> a03q.bio <- sqlFetch(a03.connect, "qryRuffeBio", na.string = "")
> comp.df(a07q.bio, a03q.bio)
```

```
[1] TRUE
```

```
> a00q.bio <- sqlFetch(a00.connect, "qryRuffeBio", na.string = "")
> comp.df(a03q.bio, a00q.bio)
```

```
[1] TRUE
```

All connections should be closed when you are done fetching data through them,

```
> odbcClose(a07.connect)
> odbcClose(a03.connect)
> odbcClose(a00.connect)
```

3.2 Excel

Reading data from Excel using the RODBC package is performed similarly to that described for Access. First, a connection to the Excel file must be created,

```
> x07.connect <- odbcConnectExcel2007("RuffeBio.xlsx") # Excel 2007
> xls.connect <- odbcConnectExcel("RuffeBio.xls") # pre-Excel 2007
```

The data “sheet” is then fetched similarly except that the name of the object is replaced by the name of the worksheet in Excel. For example, the worksheet named “Bio” in the Excel (2007) file is fetched and compared to the Access (2007) results with,

```
> x07.bio <- sqlFetch(x07.connect, "Bio", na.string = "")
> comp.df(a07q.bio, x07.bio)
```

```
[1] FALSE
```

It is seen that the Excel and Access results are not the same. A structure of the Excel results,

```
> str(x07.bio)

'data.frame':      40 obs. of  10 variables:
 $ fishID  : num  60 61 62 63 64 65 66 67 68 69 ...
 $ locShort: Factor w/ 1 level "St. Louis R. (2007)": 1 1 1 1 1 1 1 1 1 1 ...
 $ year    : num  2007 2007 2007 2007 2007 ...
 $ month   : num  9 9 9 9 9 9 9 9 9 9 ...
 $ day     : num  20 20 20 20 20 20 20 20 20 20 ...
 $ date    : Factor w/ 1 level "9/20/2007": 1 1 1 1 1 1 1 1 1 1 ...
 $ t1      : num  134 111 110 115 92 88 95 90 99 107 ...
 $ wt      : num  24.6 14.7 12.3 16 8.3 7.8 9.7 8.2 11.7 13 ...
 $ sex     : Factor w/ 3 levels "female","male",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ maturity: Factor w/ 2 levels "immature","mature": 2 2 1 2 2 2 2 2 2 2 ...
```

reveals that the `date` variable is read as a factor rather than a POSIXct object. The factor can be converted to the same POSIXct format as obtained from the Access files with,

```
> x07.bio$date <- as.POSIXct(strptime(x07.bio$date, "%m/%d/%Y", tz = ""))
> comp.df(a07q.bio, x07.bio)
```

```
[1] TRUE
```

where the second argument in `strptime` shows the format of the dates in the `xls.bio$date` input object (`%m` says the month is first, `%d` says the day is second, and `%Y` says that the four digit year is last) and the `tz=""` argument tells `strptime()` to ignore the time zone of the date.

Similar steps are used to read from the Excel (pre-2007) file,

```
> xls.bio <- sqlFetch(xls.connect, "Bio", na.string = "")
> xls.bio$date <- as.POSIXct(strptime(xls.bio$date, "%m/%d/%Y", tz = ""))
> comp.df(x07.bio, xls.bio)
```

```
[1] TRUE
```

Again, all connections should be closed when you are done fetching data through them,

```
> odbcClose(x07.connect)
> odbcClose(xls.connect)
```

An alternative approach is provided in the `xlsReadWrite` package, which is not currently available on CRAN. This package provides `read.xls()` for reading Excel (pre-2007) files². The `read.xls()` function requires only the file name as an argument if the Excel data file is constructed very much like a database table. For example, **RuffeBio.xls** can be read with,

```
> library(xlsReadWrite)
> xls.bio.2 <- read.xls("RuffeBio.xls")
> xls.bio.2$date <- as.POSIXct(strptime(xls.bio.2$date, "%m/%d/%Y", tz = ""))
> str(xls.bio.2)
```

```
'data.frame':      40 obs. of  10 variables:
 $ fishID  : num  60 61 62 63 64 65 66 67 68 69 ...
 $ locShort: Factor w/ 1 level "St. Louis R. (2007)": 1 1 1 1 1 1 1 1 1 1 ...
 $ year    : num  2007 2007 2007 2007 2007 ...
 $ month   : num  9 9 9 9 9 9 9 9 9 ...
 $ day     : num  20 20 20 20 20 20 20 20 20 ...
 $ date    : POSIXct, format: "2007-09-20" "2007-09-20" ...
 $ tl      : num  134 111 110 115 92 88 95 90 99 107 ...
 $ wt      : num  24.6 14.7 12.3 16 8.3 7.8 9.7 8.2 11.7 13 ...
 $ sex     : Factor w/ 3 levels "female","male",...: 1 1 1 1 1 1 1 1 1 ...
 $ maturity: Factor w/ 3 levels "", "immature",...: 3 3 2 3 3 3 3 3 3 ...
```

The `read.xls()` function does NOT have a `na.string=` argument and, thus, the missing values in a character or factor variable cannot be handled upon input. The blank cells can be set to `NA` with the following commands,

```
> is.na(xls.bio.2$maturity) <- which(xls.bio.2$maturity == "")
> xls.bio.2$maturity <- factor(xls.bio.2$maturity)
> comp.df(x07.bio, xls.bio.2)

[1] TRUE
```

3.2.1 “Odd”-shaped Excel Files

Problems can arise when Excel data files are not so strictly like a database table (variable names in first row, data starts in the second row, very few missing fields, etc.; as the **RuffeBio.xls** file is). For example, the **RuffeBio2.xls** file has variable names that start in the fourth row, a non-data title in the first cell (i.e., A1), and `tl` and `maturity` variables that have missing values in the first 26 rows.

The RODBC methodology produces the following result with the **RuffeBio2.xls** file,

```
> xls2.connect <- odbcConnectExcel("RuffeBio2.xls")
> xls2.bio <- sqlFetch(xls2.connect, "Bio", na.string = "")
> str(xls2.bio)

'data.frame':      43 obs. of  10 variables:
 $ Example of an odd-shaped Excel file: num  NA NA NA 60 61 62 63 64 65 66 ...
 $ F2                                     : Factor w/ 2 levels "locShort","St. Louis R. (2007)": NA NA ...
```

²The `gdata` package, also available on CRAN, also contains a `read.xls()` function. This function works by converting the Excel file to a comma-separated-values (CSV) file and then reading that file into R (see Section 4). This requires a version of PERL on your system. It should be noted that the `gdata` package is loaded with the `FSA` package.

```

$ F3 : num NA NA NA 2007 2007 ...
$ F4 : num NA NA NA 9 9 9 9 9 ...
$ F5 : num NA NA NA 20 20 20 20 20 ...
$ F6 : Factor w/ 2 levels "9/20/2007","date": NA NA 2 1 1 1 1 1 ...
$ F7 : Factor w/ 1 level "t1": NA NA 1 NA NA NA NA NA NA ...
$ F8 : num NA NA NA 24.6 14.7 12.3 16 8.3 7.8 9.7 ...
$ F9 : Factor w/ 4 levels "female","male",,..: NA NA 3 1 1 1 1 1 ...
$ F10 : Factor w/ 3 levels "immature","mature",,..: NA NA 3 NA NA NA ...

```

```
> odbcClose(xls2.connect)
```

Clearly, the RODBC method does not work well with Excel files that do not start in the top row³.

In situations like these, `read.xls()` will work better, but it requires additional arguments. First, `read.xls()` can be told, in the `from=` argument, which row to start reading the data from. Second, if `read.xls()` cannot tell in the first few rows of a column what type of variable is in that column then it will default to calling the variable in that column a logical variable. In the **RuffeBio2.xls** file, the `t1` variable had missing data in the first 26 rows and `read.xls()` will not be able to determine what type of variable is in this column. This particular problem can be remedied by specifically telling `read.xls()`, in the `colClasses=` argument, what type of variable class is in each column. With these two modifications, the **RuffeBio2.xls** file can be properly loaded into R with,

```

> xls2.bio.1 <- read.xls("RuffeBio2.xls", from = 4, colClasses = c("numeric",
+   "factor", "numeric", "numeric", "numeric", "character", "numeric",
+   "numeric", "factor", "factor"))
> xls2.bio.1$date <- as.POSIXct(strptime(xls2.bio.1$date, "%m/%d/%Y",
+   tz = ""))
> is.na(xls2.bio.1$maturity) <- which(xls2.bio.1$maturity == "")
> xls2.bio.1$maturity <- factor(xls2.bio.1$maturity)
> str(xls2.bio.1)

```

```

'data.frame':      40 obs. of  10 variables:
 $ fishID : num  60 61 62 63 64 65 66 67 68 69 ...
 $ locShort: Factor w/ 1 level "St. Louis R. (2007)": 1 1 1 1 1 1 1 1 1 1 ...
 $ year   : num  2007 2007 2007 2007 2007 ...
 $ month  : num  9 9 9 9 9 9 9 9 9 9 ...
 $ day    : num  20 20 20 20 20 20 20 20 20 20 ...
 $ date   : POSIXct, format: "2007-09-20" "2007-09-20" ...
 $ t1     : num  NA NA NA NA NA NA NA NA NA NA ...
 $ wt     : num  24.6 14.7 12.3 16 8.3 7.8 9.7 8.2 11.7 13 ...
 $ sex    : Factor w/ 3 levels "female","male",,..: 1 1 1 1 1 1 1 1 1 1 ...
 $ maturity: Factor w/ 2 levels "immature","mature": NA NA NA NA NA NA NA NA NA NA ...

```

4 Reading from Text Files

Tab-delimited text files can be read into R with `read.table()`. The `read.table()` function requires only one argument – the name of the file in quotes. However, if the first row of the data file contains the variable names, as **RuffeBio.txt** does, then R must be told so with the `header=TRUE` argument. By default, `read.table()` simply looks for fields that are separated by “white space” whether that be a space, multiple spaces, or a tab. There are at least two problems with this approach. First, if the file contains missing cells then those missing cells appear as spaces in the file and `read.table()` will lump those spaces

³It should be noted, though, that the RODBC method will “find” the proper first row in the worksheet as long as all of the rows above it are blank

with the tabs before and after the missing field to make it look like a single field delimiter. In this case, `read.table()` will return an error saying “line XX did not have YY elements”. Second, if any one of the field entries contains spaces then each “word” in that field will be incorrectly considered as a separate field. For example, the `locShort` field contains the text “St. Louis R. (2007)”. The default `read.table()` will treat this single entry as four entries and will result in an error saying “more columns than column names”. The easiest way to address these errors is to force `read.table()` to delimit fields with tabs by using the `sep=` argument. Thus, **RuffeBio.txt** can be loaded into R and compared to the Excel data framewith,

```
> txt.bio <- read.table("RuffeBio.txt", header = TRUE, sep = "\t", na.strings = "")
> txt.bio$date <- as.POSIXct(strptime(txt.bio$date, "%m/%d/%Y", tz = ""))
> str(txt.bio)

'data.frame':      40 obs. of  10 variables:
 $ fishID  : int  60 61 62 63 64 65 66 67 68 69 ...
 $ locShort: Factor w/ 1 level "St. Louis R. (2007)": 1 1 1 1 1 1 1 1 1 1 ...
 $ year    : int  2007 2007 2007 2007 2007 2007 2007 2007 2007 2007 ...
 $ month   : int   9 9 9 9 9 9 9 9 9 9 ...
 $ day     : int  20 20 20 20 20 20 20 20 20 20 ...
 $ date    : POSIXct, format: "2007-09-20" "2007-09-20" ...
 $ tl      : int  134 111 110 115 92 88 95 90 99 107 ...
 $ wt      : num  24.6 14.7 12.3 16 8.3 7.8 9.7 8.2 11.7 13 ...
 $ sex     : Factor w/ 3 levels "female","male",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ maturity: Factor w/ 2 levels "immature","mature": 2 2 1 2 2 2 2 2 2 2 ...
```

Note that the only difference between this data frame and the previous data frame is that several of the variables are considered to be integer rather than numeric types.

Comma-separated-values (CSV) are text files where each field is separated by commas. Comma-separated-values files can be loaded into R with `read.csv()` with the only required argument being the name of the data file in quotes. It should also be noted that `read.csv()`, in contrast to `read.table()`, defaults to using `header=TRUE`. Thus, **RuffeBio.csv** is loaded into R with,

```
> csv.bio <- read.csv("RuffeBio.csv", na.strings = "")
> csv.bio$date <- as.POSIXct(strptime(csv.bio$date, "%m/%d/%Y", tz = ""))
> comp.df(txt.bio, csv.bio)
```

```
[1] TRUE
```

5 Reading Data Files in R Packages

In many texts and vignettes one will see data loaded into R with the `data()` function. This function is used explicitly to load data files that are parts of R packages. For example, the **Alewifelh** from the **FSAdata** package can be loaded with,

```
> library(FSAdata)

> data(Alewifelh)
> str(Alewifelh)

'data.frame':      104 obs. of  2 variables:
 $ otoliths: int  0 0 1 1 1 1 1 1 1 1 ...
 $ scales  : int  0 0 0 1 1 1 1 1 1 1 ...
```

6 Creating Subsets of Data Frames

It is very common that a researcher will want to examine a subset of a larger data frame – e.g., examine just male ruffe. R provides the `subset()` function for creating subsets of data frames. For example, the male ruffe can be extracted from any of the previous data frames with (the details will be explained later),

```
> r.male <- subset(txt.bio, sex == "male")
```

and a frequency table of `sex` constructed with,

```
> table(r.male$sex)

female  male unknown
      0     8      0
```

It is immediately obvious from this table that `subset()` does indeed extract just the males but the level names of the `sex` variable still contains the “female” and “unknown” levels. This produces tables (and graphs) that are “ugly.” There are a variety of methods for correcting this problem but the easiest is to use `Subset()` from the `NCStats` package⁴.

The `Subset()` function requires the original data frame as the first argument and a conditioning statement as the second argument. The conditioning statement is a statement that is used to either include or exclude the individuals from the original data frame that will make up the new data frame. The result from `Subset()` should be stored in a new object which will then be the name of the new data frame. The conditioning statements used in `Subset()` can be fairly complex (**Table 1**). For example,

```
> library(NCStats)

> r.male <- Subset(txt.bio, sex == "male")
> table(r.male$sex)

male
  8
```

Table 1. Condition operators used in `Subset()` and their results. Note that *variable* generically represents a variable in the original data frame and *value* is a generic value or level. Both of these would be replaced with specific items.

Comparison Operator	Individuals Returned from Original Data Frame
<i>variable</i> == <i>value</i>	all individual equal to given value
<i>variable</i> != <i>value</i>	all individuals NOT equal to given value
<i>variable</i> > <i>value</i>	all individuals greater than given value
<i>variable</i> >= <i>value</i>	all individuals greater than or equal to given value
<i>variable</i> < <i>value</i>	all individuals less than given value
<i>variable</i> <= <i>value</i>	all individuals less than given value
<i>condition</i> & <i>condition</i>	all individuals that meet both conditions
<i>condition</i> <i>condition</i>	all individuals that meet one or both conditions

The following items are examples of new data frames created by subsetting the `txt.bio` data frame⁵.

⁴This package is loaded with `FSA` .

⁵The `view()` function is used in the examples below to show a random selection of rows in the new data frame.

- A data frame that contains only males.

```
> r.male <- Subset(txt.bio, sex == "male")
> view(r.male)
```

fishID	locShort	year	month	day	date	tl	wt	sex	maturity	
18	77 St. Louis R.	(2007)	2007	9	20	2007-09-20	NA	13.1	male	mature
27	86 St. Louis R.	(2007)	2007	9	20	2007-09-20	84	5.7	male	mature
32	91 St. Louis R.	(2007)	2007	9	20	2007-09-20	99	9.1	male	mature
33	92 St. Louis R.	(2007)	2007	9	20	2007-09-20	84	5.6	male	mature
36	95 St. Louis R.	(2007)	2007	9	20	2007-09-20	81	5.8	male	mature
39	98 St. Louis R.	(2007)	2007	9	20	2007-09-20	NA	10.5	male	mature

- A data frame that contains *male* and *females* (but not *unknown* sex individuals).

```
> r.fm1 <- Subset(txt.bio, sex == "male" | sex == "female")
> view(r.fm1)
```

fishID	locShort	year	month	day	date	tl	wt	sex	maturity	
28	87 St. Louis R.	(2007)	2007	9	20	2007-09-20	105	11.7	male	mature
31	90 St. Louis R.	(2007)	2007	9	20	2007-09-20	102	9.5	female	mature
33	92 St. Louis R.	(2007)	2007	9	20	2007-09-20	84	5.6	male	mature
36	95 St. Louis R.	(2007)	2007	9	20	2007-09-20	81	5.8	male	mature
37	96 St. Louis R.	(2007)	2007	9	20	2007-09-20	65	2.9	female	mature
40	99 St. Louis R.	(2007)	2007	9	20	2007-09-20	115	17.9	female	mature

```
> r.fm2 <- Subset(txt.bio, sex != "unknown")
> view(r.fm1)
```

fishID	locShort	year	month	day	date	tl	wt	sex	maturity	
3	62 St. Louis R.	(2007)	2007	9	20	2007-09-20	110	12.3	female	immature
4	63 St. Louis R.	(2007)	2007	9	20	2007-09-20	115	16.0	female	mature
14	73 St. Louis R.	(2007)	2007	9	20	2007-09-20	105	10.2	female	mature
18	77 St. Louis R.	(2007)	2007	9	20	2007-09-20	NA	13.1	male	mature
30	89 St. Louis R.	(2007)	2007	9	20	2007-09-20	104	11.2	female	mature
33	92 St. Louis R.	(2007)	2007	9	20	2007-09-20	84	5.6	male	mature

- A data frame that contains individuals with a total length greater than 80 mm.

```
> r.gt80 <- Subset(txt.bio, tl > 80)
> view(r.gt80)
```

fishID	locShort	year	month	day	date	tl	wt	sex	maturity	
9	68 St. Louis R.	(2007)	2007	9	20	2007-09-20	99	11.7	female	mature
13	72 St. Louis R.	(2007)	2007	9	20	2007-09-20	102	11.4	female	mature
14	73 St. Louis R.	(2007)	2007	9	20	2007-09-20	105	10.2	female	mature
24	83 St. Louis R.	(2007)	2007	9	20	2007-09-20	111	16.5	female	mature
30	89 St. Louis R.	(2007)	2007	9	20	2007-09-20	104	11.2	female	mature
35	94 St. Louis R.	(2007)	2007	9	20	2007-09-20	81	5.8	female	mature

- A data frame that contains males with a total length greater than 80 mm.

```
> r.mgt80 <- Subset(txt.bio, sex == "male" & tl > 80)
> view(r.mgt80)
```

fishID	locShort	year	month	day	date	tl	wt	sex	maturity	
27	86 St. Louis R.	(2007)	2007	9	20	2007-09-20	84	5.7	male	mature
28	87 St. Louis R.	(2007)	2007	9	20	2007-09-20	105	11.7	male	mature
32	91 St. Louis R.	(2007)	2007	9	20	2007-09-20	99	9.1	male	mature
33	92 St. Louis R.	(2007)	2007	9	20	2007-09-20	84	5.6	male	mature
34	93 St. Louis R.	(2007)	2007	9	20	2007-09-20	87	7.6	male	mature
36	95 St. Louis R.	(2007)	2007	9	20	2007-09-20	81	5.8	male	mature

Note that after each subsetting you should get in the habit of either typing the name of the new data frame or putting the new data frame into the `head()` or `view()` functions. While this is not required it is highly recommended as this is a way for you to determine if the data frame actually contains the items that you desire.

7 Final Comments on Data Entry

My recommendation is to keep your data in its native format if at all possible. By this I mean if your data is in Access then keep it in Access rather than converting it to Excel, a CSV, or a tab-delimited text file. Similarly, if your data is in Excel then try to read it from Excel rather than converting to CSV or tab-delimited text file. I make this recommendation primarily because it will be much easier on you if you have to modify the data file and re-read it into R. For example, if you data is in Access but you converted it to a text file then if you modify Access you will have to make sure to save a new version of the text file. In contrast, if you are reading directly from the Access file then you simply need to re-read the modified file.

I have focused this presentation on reading data from Microsoft products not because I want to promote Microsoft products but because most questions that I receive are related to these products. As an open-source product, R contains other functions for reading data from Oracle files (see XXX) and MySQL files (see XXX). In addition, functions have been constructed to access Open Office Calc spreadsheets (see XXX).