# fishR Vignette - Data Entry

Dr. Derek Ogle, Northland College March 2, 2013

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 tab-delimited, comma-separated values (CSV), MSExcel (2007; .xlsx), MSExcel (pre-2007 .xls), MSAccess (2007), and MSAccess (pre-2007) files. In addition, I will briefly mention 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. It should also be noted that I work primarily on a Windows-based machine and have not tested the methods below on Unix- or Mac-based machines, though I have tried to comment where I think that what is being described is Windows specific.

The code below relies on the `FSA` package maintained by the author as well as the `XLConnect` and `RODBC` packages maintained by other authors. These packages are loaded with

```
> library(FSA)
> library(XLConnect)
> library(RODBC)
> library(reshape2)
```

## 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 FishR General Examples web 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. 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()`. In general, this interactive method should be avoided as it will require the user to interact with the program whenever an R script is run, rather than having the script run to completion without additional interaction by the user.

# 2 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 **SMBassWB** from the `FSA` package can be loaded with

```
> data(SMBassWB)
> str(SMBassWB)
'data.frame': 445 obs. of  20 variables:
 $ species: Factor w/ 1 level "SMB": 1 1 1 1 1 1 1 1 1 1 ...
 $ lake   : Factor w/ 1 level "WB": 1 1 1 1 1 1 1 1 1 1 ...
```

```
$ gear   : Factor w/ 2 levels "E","T": 1 1 1 1 1 1 1 1 1 1 ...
$ yearcap: int  1988 1988 1988 1988 1988 1988 1989 1990 1990 1990 ...
$ fish   : int  5 3 2 4 6 7 50 482 768 428 ...
$ agecap : int  1 1 1 1 1 1 1 1 1 1 ...
$ lencap : int  71 64 57 68 72 80 55 75 75 71 ...
$ anu1   : num  1.91 1.88 1.09 1.32 1.59 ...
$ anu2   : num  NA NA NA NA NA NA NA NA NA NA ...
$ anu3   : num  NA NA NA NA NA NA NA NA NA NA ...
$ anu4   : num  NA NA NA NA NA NA NA NA NA NA ...
$ anu5   : num  NA NA NA NA NA NA NA NA NA NA ...
$ anu6   : num  NA NA NA NA NA NA NA NA NA NA ...
$ anu7   : num  NA NA NA NA NA NA NA NA NA NA ...
$ anu8   : num  NA NA NA NA NA NA NA NA NA NA ...
$ anu9   : num  NA NA NA NA NA NA NA NA NA NA ...
$ anu10  : num  NA NA NA NA NA NA NA NA NA NA ...
$ anu11  : num  NA NA NA NA NA NA NA NA NA NA ...
$ anu12  : num  NA NA NA NA NA NA NA NA NA NA ...
$ radcap : num  1.91 1.88 1.1 1.33 1.59 ...
```

Of course, this is only useful for data that is likely not yours and is found in an R package.

# 3 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 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="\t"` argument. Additionally, one can identify a string that when read is considered to be a missing value. The default behavior of `read.table()` is to consider `NA` as a missing value. However, it is more common that missing values will truly be missing and, in this case, one should use `na.strings=""`. Thus, **RuffeBio.txt** can be loaded into R, the *date* variable format converted[1], and the data frame structure observed with

```
> 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" ...
```

[1]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.

```
 $ 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 ...
```

Comma-separated-values (CSV) files are text files where each field is separated by commas. Commas-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 and the data variable format is converted with

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

# 4 Reading from Excel Files

## 4.1 using XLConnect

The `XLConnect` package provides useful and flexible functions for reading from and writing to Excel files. In this section, I will describe only simple methods of reading data from Excel files. The excellent demo vignette provided with the `XLConnect` package should be consulted for more advanced uses.

The Excel file, or workbook, must first be "loaded" into R before specific worksheets can be read from it. This is accomplished by providing the name of the file in the first argument of `loadWorkbook()`. It is important to save the results of this function to an object. For example, the Excel 2007 example file is loaded with

```
> x07.wb <- loadWorkbook("RuffeBio.xlsx")    # Excel 2007
```

Once a file is loaded into R, a worksheet inside that file can be read with `readWorksheet()`. This function requires the `loadWorkbook` object as the first argument and the name of worksheet to be read in quotes in the `sheet=` argument. Thus, the **Bio** worksheet from the **RuffeBio.xlsx** file is read and the structure is obsered with

```
> x07.bio <- readWorksheet(x07.wb,sheet="Bio")
> str(x07.bio)
'data.frame': 40 obs. of  10 variables:
 $ fishID  : num  60 61 62 63 64 65 66 67 68 69 ...
 $ locShort: chr  "St. Louis R. (2007)" "St. Louis R. (2007)" "St. Louis R. (2007)" "St. Louis R. (2
```

```
 $ 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    : chr  "9/20/2007" "9/20/2007" "9/20/2007" "9/20/2007" ...
 $ 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     : chr  "female" "female" "female" "female" ...
 $ maturity: chr  "mature" "mature" "immature" "mature" ...
```

It is seen that `readWorkseet()` does not convert character strings to factors as `read.table()` does. The *date* variable can be converted to a proper date format as shown previously. The other variables can be converted to factor variables with `factor()`. However, after doing so it will be apparent that the "missing data" in the *maturity* variable will be treated as a level. To rectify this, the missing values should be converted to `NA`s before factoring. All of these conversions are illustrated with

```
> x07.bio$date <- as.POSIXct(strptime(x07.bio$date,"%m/%d/%Y",tz=""))
> x07.bio$locShort <- factor(x07.bio$locShort)
> x07.bio$sex <- factor(x07.bio$sex)
> x07.bio$maturity[x07.bio$maturity==""] <- NA
> x07.bio$maturity <- factor(x07.bio$maturity,exclude="")
> 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    : 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","mature",..: 2 2 1 2 2 2 2 2 2 2 ...
```

With these changes, this file matches the tab-delimited text file read previously with the exception that some variables are recorded as "numerical" rather than "integer."

A similar process is followed for pre-2007 versions of Excel (demonstrated in the next paragraph).

Some Excel data files do not so strictly resemble a database table (i.e., they don't have variable names in the first row, data starting in the second row after the labels, very few missing fields, etc.) as the **RuffeBio.xls** file does. 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. These types of files can be adequately read with `readWorksheet()` using the `startRow=` argument to identify which row to start reading from[2]. For example, the **RuffeBio2.xls** is read with

```
> xodd.wb <- loadWorkbook("RuffeBio2.xls")
> xodd.bio <- readWorksheet(xodd.wb,sheet="Bio",startRow=4)
> str(xodd.bio)
'data.frame': 40 obs. of  10 variables:
 $ fishID  : num  60 61 62 63 64 65 66 67 68 69 ...
 $ locShort: chr  "St. Louis R. (2007)" "St. Louis R. (2007)" "St. Louis R. (2007)" "St. Louis R. (2
 $ year    : num  2007 2007 2007 2007 2007 ...
```

[2] Also note that there are `endRow=`, `startCol=`, and `endCol=` arguments if one must read a range of data that does not start in the upper-right cell of the worksheet.

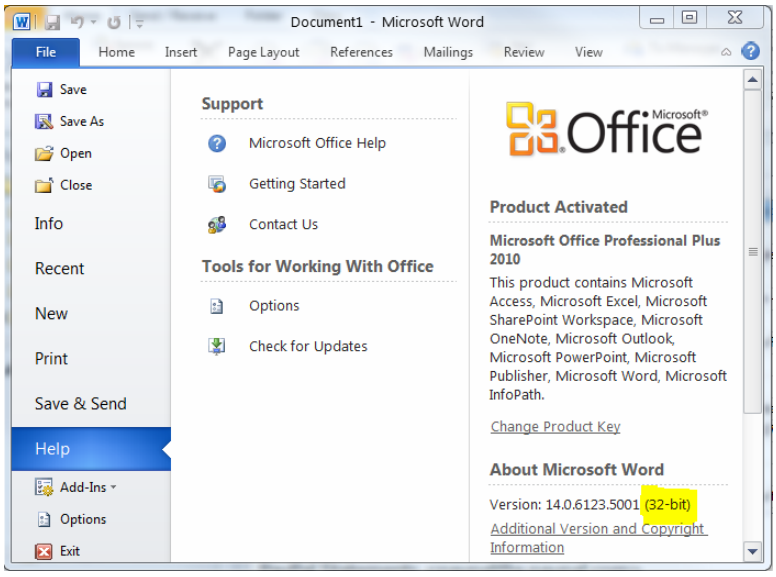
```
$ month   : num  9 9 9 9 9 9 9 9 9 9 ...
$ day     : num  20 20 20 20 20 20 20 20 20 20 ...
$ date    : chr  "9/20/2007" "9/20/2007" "9/20/2007" "9/20/2007" ...
$ tl      : num  NA NA NA NA NA NA NA NA NA NA ...
$ wt      : num  24.6 14.7 12.3 16 8.3 7.8 9.7 8.2 11.7 13 ...
$ sex     : chr  "female" "female" "female" "female" ...
$ maturity: chr  NA NA NA NA ...
```

## 4.2 Using RODBC

The `RODBC` packag provides functions for which a user can import data directly from external files, including Excel and Access, into R. With the advent of 64-bit versions of R, Microsoft products, and computers, one has to be much more careful using the RODBC functions than previously. In particular, one must make sure that your workflow is *using the same architecture for R as it is for the Microsoft products* if one is going to successfully read directly from those products into R. In other words, R and Microsoft Office both either need to be 32-bit or 64-bit, but not a mixture of both (e.g., you can't use 64-bit R with 32-bit Office). One can determine their version of R with `version`, or more directly with `version$platform` as such,

```
> version$platform
[1] "i386-w64-mingw32"
```

If one sees "i386" and "mingw32" in this then you are using the 32-bit version of R. The version of your Microsoft Office product can generally be found in the "Help..About" or "File .. Help" information. For example, on my current machine, I am using a 32-bit version of Office (see the lower-right highlighted item below)




Thus, I should be able to successfully use `RODBC` as described below because both my R and Microsoft Office are using the 32-bit architecture.

Reading data from Excel using the `RODBC` package is very similar to what was described above for using `XLConnect`. First, a connection to the Excel file must be created with either `odbcConnectExcel2007()` for Excel (2007) files or `odbcConnectExcel()` for pre-Excel (2007) files. For example,

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

The worksheet in one of these files is then "fetched" with `sqlFetch()` which requires two arguments. The first argument is the saved "connection" name that was just created. The second argument is the name, in quotes, of the worksheet on the connected Excel file that contains the data to be fetched. Also, notice that true missing values will be considered as R missing values (i.e., as `NA`s) when `na.strings=""` is used. For example, the worksheet named "Bio" in the Excel (2007) file is fetched and the structure is obsered with

```
> x07.bio2 <- sqlFetch(x07.con,"Bio",na.strings="")
> str(x07.bio2)
'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 ...
 $ 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/ 2 levels "immature","mature": 2 2 1 2 2 2 2 2 2 2 ...
```

It is seen that RODBC reads the *date* variable as a factor rather than a character or POSIXct object. This conversion is accomplished as preiously with

```
> x07.bio2$date <- as.POSIXct(strptime(x07.bio2$date,"%m/%d/%Y",tz=""))
```

Similar steps are used to read from the Excel (pre-2007) file (making sure to use the `xls.con` connection from above). All connections should be closed when you are done fetching data through them. This is accomplished with `odbcClose()` as shown below

```
> odbcClose(x07.con)
> ## odbcClose(xls.con)
```

The RODBC method can NOT be reliably used for Excel files that do not closely look like a database table (e.g., like **RuffeBio2.xls**). For example, examine the structure of the data frame produced with

```
> xls2.con <- odbcConnectExcel("RuffeBio2.xls")
> xls2.bio <- sqlFetch(xls2.con,"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 1
 $ F3                                 : num  NA NA NA 2007 2007 ...
 $ F4                                 : num  NA NA NA 9 9 9 9 9 9 9 ...
 $ F5                                 : num  NA NA NA 20 20 20 20 20 20 20 ...
 $ F6                                 : Factor w/ 2 levels "9/20/2007","date": NA NA 2 1 1 1 1 1 1 1
 $ F7                                 : Factor w/ 1 level "tl": NA NA 1 NA 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 1 1
 $ F10                                : Factor w/ 3 levels "immature","mature",..: NA NA 3 NA NA NA
> odbcClose(xls2.con)
```

# 5 Access

The `RODBC` package can also be used to directly port from Access to R. The functions are very similar to those described for Excel in Section 4.2. In addition, if you did not read Section 4.2, then you should go there now to read the discussion and warning related to the 32- and 64-bit architectures of R and Microsoft Office.

A channel of communication must first be opened between R and the application which holds the data using one of `odbcConnectAccess2007()` for Access (2007) files or `odbcConnectAccess()` for pre-Access (2007) files. Each of these is shown below, though, of course, you would only use one of these commands that was specific to the data file you wish to use.

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

Once the connection to the data file has been created then the data is "fetched" with `sqlFetch()` where the first argument is the name of the connection created above and 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 the structure observed with

```
> a07d.bio <- sqlFetch(a07.con,"dbRuffeBio",na.strings="")
> 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/ 2 levels "immature","mature": 2 2 1 2 2 2 2 2 2 2 ...
```

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

```
> a07q.bio <- sqlFetch(a07.con,"qryRuffeBio",na.string="")
> str(a07q.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/ 2 levels "immature","mature": 2 2 1 2 2 2 2 2 2 2 ...
```

Similar commands can be used to read from the Access (2003) and Access (2000) files (simply making sure to use the correct connection created above). All connections should be closed when you are done fetching data through them with

```
> odbcClose(a07.con)
> ## odbcClose(a03.con)
> ## odbcClose(a00.con)
```

# 6 Final Comments on Reading Data into R

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 the Access file, 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 ROracle), mySQL files (see RMySQL), and Open Office Base files (see ODB).

# 7 Creating Subsets of Data Frames

It is common that a researcher will want to examine a subset of a larger data frame – e.g., examine just male ruffe. Base R provides `subset()` 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 *sex* variable still contains the "female" and "unknown" level names. 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 `FSA` package.

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 a new data frame. The conditioning statements used in `Subset()` can be fairly complex (Table 1).

As an example, the subset of just male ruffe is found using `Subset()`, with a table of the result shown to demonstrate that the "female" level name has indeed been removed, with

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

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 |
|---|---|
| *variable* == *value* | all individual equal to given value |
| *variable* != *value* | all individuals NOT equal to given value |
| *variable* > *value* | all individuals greater than given value |
| *variable* >= *value* | all individuals greater than or equal to given value |
| *variable* < *value* | all individuals less than given value |
| *variable* <= *value* | all individuals less than given value |
| *condition* & *condition* | all individuals that meet both conditions |
| *condition* \| *condition* | all individuals that meet one or both conditions |

```
male
   8
```

The following items are examples of new data frames created by subsetting the *txt.bio* data frame[3].

- 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
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
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 *female*s (but not *unknown* sex individuals).

```
> r.fm1 <- Subset(txt.bio,sex=="male" | sex=="female")  # male OR female
> view(r.fm1)
   fishID            locShort year month day       date  tl   wt    sex maturity
1      60 St. Louis R. (2007) 2007     9  20 2007-09-20 134 24.6 female   mature
2      61 St. Louis R. (2007) 2007     9  20 2007-09-20 111 14.7 female   mature
5      64 St. Louis R. (2007) 2007     9  20 2007-09-20  92  8.3 female   mature
8      67 St. Louis R. (2007) 2007     9  20 2007-09-20  90  8.2 female   mature
19     78 St. Louis R. (2007) 2007     9  20 2007-09-20  56  2.3 female immature
23     82 St. Louis R. (2007) 2007     9  20 2007-09-20 110 14.4 female   mature
> r.fm2 <- Subset(txt.bio,sex!="unknown")               # exclude unknowns
> view(r.fm2)
   fishID            locShort year month day       date  tl   wt    sex maturity
6      65 St. Louis R. (2007) 2007     9  20 2007-09-20  88  7.8 female   mature
8      67 St. Louis R. (2007) 2007     9  20 2007-09-20  90  8.2 female   mature
15     74 St. Louis R. (2007) 2007     9  20 2007-09-20  90  7.6 female   mature
```

[3]The `view()` function is used in the examples below to show a random selection of six rows from the new data frame.

```
17     76 St. Louis R. (2007) 2007     9  20 2007-09-20 114 12.4 female   mature
31     90 St. Louis R. (2007) 2007     9  20 2007-09-20 102  9.5 female   mature
39     98 St. Louis R. (2007) 2007     9  20 2007-09-20  NA 10.5   male   mature
```

```
> r.fm3 <- Subset(txt.bio,sex%in%c("male","female"))
> view(r.fm3)
```

```
   fishID            locShort year month day       date  tl   wt    sex maturity
2      61 St. Louis R. (2007) 2007     9  20 2007-09-20 111 14.7 female   mature
4      63 St. Louis R. (2007) 2007     9  20 2007-09-20 115 16.0 female   mature
13     72 St. Louis R. (2007) 2007     9  20 2007-09-20 102 11.4 female   mature
15     74 St. Louis R. (2007) 2007     9  20 2007-09-20  90  7.6 female   mature
29     88 St. Louis R. (2007) 2007     9  20 2007-09-20 120 18.6 female   mature
39     98 St. Louis R. (2007) 2007     9  20 2007-09-20  NA 10.5   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
6      65 St. Louis R. (2007) 2007     9  20 2007-09-20  88  7.8 female   mature
8      67 St. Louis R. (2007) 2007     9  20 2007-09-20  90  8.2 female   mature
16     75 St. Louis R. (2007) 2007     9  20 2007-09-20 102 11.2 female   mature
28     87 St. Louis R. (2007) 2007     9  20 2007-09-20 105 11.7   male   mature
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
```

- 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 to see all of the resulting data, using `head()` or `view()` to examined a subsample of rows from the new data frame, or using `str()` to examine the structure of the new data frame. While this practice is not required, it is highly recommended as a way to determine if the new data frame actually contains the items that you desire.

# 8 Changing "Directions" of the Data

Suppose you have data that consists of the counts of several species of fish from several sets of nets. These type of data may be entered in one of two formats. In the so-called "wide" format, each row of the data frame would consist of one net set and there would be multiple columns corresponding to each species of fish. This type of data would look like this

```
  net eff temp BKT LKT RBT
1   1   1 17.0  17  45  14
2   2   1 17.3   5   0  13
3   3   1 17.2   0  17  17
```

where `net` is a net identification number; `BKT`, `LKT`, and `RBT` represent the various species of fish captured; and `eff` and `temp` are "effort" and water temperature, respectively, and are meant to illustrate the type of "net-specific" data that might be collected in addition to the catch of individual species. In so-called "long" format each row does NOT correspond to a net, rather each row corresponds to a net-species combination. Thus, in long format the catches of all species in one net are distributed across several rows. The same data shown above in wide format is shown below in long format

```
  net eff temp species catch
1   1   1 17.0     BKT    17
2   2   1 17.3     BKT     5
3   3   1 17.2     BKT     0
4   1   1 17.0     LKT    45
5   2   1 17.3     LKT     0
6   3   1 17.2     LKT    17
7   1   1 17.0     RBT    14
8   2   1 17.3     RBT    13
9   3   1 17.2     RBT    17
```

It is fairly common to need to convert data in "long" format to that in "wide" format or vice versa. Methods for making these changes are described in the following subsections. For all examples below, suppose that the wide data above is stored in `dfw` and the long data above is stored in `dfl`.

## 8.1 Using `reshape()` from Base R

The `reshape()` function in base R can be used to convert between the two formats. In either conversion, the first argument to `reshape()` is the name of the data frame to be reshaped. The `direction=` argument can be set either to `"wide"` or `"long"` depending on the desired direction of the RESULTING data frame. To convert from wide to long format one would also use the following arguments:

- `idvar=`: The variable in wide format that represents the individuals or unit of measurements.
- `varying=`: The variable(s) in wide format that represent the "multiple measurements" on the individuals. These variables will ultimately be stacked into a single variable in the resulting long format.
- `v.names=`: This is a name for the new variable in the long format that is to be constructed from the `varying=` variables from the wide format.
- `timevar=`: This is a name for the new variable in the long format that is constructed so as to keep track of the original "group" of the multiple measurements from the wide format.
- `times=`: This is a vector of the "group" names to be used in the variable declared in `timevar=`. Often times this will be very closely related to the names given in `varying=`.

It should be noted that the variables not listed in `idvar=` or `varying=` will essentially be treated as constant variables that are closely linked to the variable in `idvar=`. Thus, the data shown above in wide format can be converted to long format with

```
> ( dfL <- reshape(dfw,direction="long",idvar="net",
                   varying=c("BKT","LKT","RBT"),v.names="catch",
                   timevar="species",times=c("BKT","LKT","RBT")) )
```

```
      net eff temp species catch
1.BKT   1   1 17.0     BKT    17
2.BKT   2   1 17.3     BKT     5
3.BKT   3   1 17.2     BKT     0
1.LKT   1   1 17.0     LKT    45
2.LKT   2   1 17.3     LKT     0
3.LKT   3   1 17.2     LKT    17
1.RBT   1   1 17.0     RBT    14
2.RBT   2   1 17.3     RBT    13
3.RBT   3   1 17.2     RBT    17
```

The same function can also be used to convert from long to wide format. In this case, the following arguments are used

- `idvar=`: The variable in long format that represents the individuals or unit of measurements.
- `v.names=`: The variable in long format that represents the multiple measurements.
- `timevar=`: The variable in long format that represents the variable that identifies the "groups" on which the multiple measurements were made.

The variables not listed in `idvar=`, `v.names=`, or `timevar=` will be treated as constant variables that are closely linked to the variable in `idvar=`. Thus, the data shown above in long format can be converted to wide format with

```
> ( dfW <- reshape(dfl,direction="wide",idvar="net",v.names="catch",timevar="species") )
  net eff temp catch.BKT catch.LKT catch.RBT
1   1   1 17.0        17        45        14
2   2   1 17.3         5         0        13
3   3   1 17.2         0        17        17
```

## 8.2 Using `cast()` and `melt()` from `reshape2`

The `reshape2` package offers alternatives to using `reshape()` from base R. To convert from a wide format to a long format the `melt()` function is used. As above, the first argument to this function is the wide format data frame to be "melt"ed. The following other arguments are used

- `id.vars=`: The variable in wide format that represents the individuals or unit of measurements.
- `measure.vars=`: The variable(s) in wide format that represent the "multiple measurements" on the individuals. These variables will ultimately be stacked into a single variable in the resulting long format.
- `variable.name=`: This is a name for the new variable in the long format that is constructed so as to keep track of the original "group" of the multiple measurements in the wide format.
- `value.name=`: This is a name for the new variable in the long format that is to be constructed from the `varying=` variables from the wide format.

The data shown above in wide format can be converted to long format with

```
> dfL <- melt(dfw,id.vars=c("net","eff","temp"),measure.vars=c('BKT','LKT','RBT'),
              variable.name="species",value.name="catch")
```

```
  net eff temp species catch
1   1   1 17.0     BKT    17
2   2   1 17.3     BKT     5
3   3   1 17.2     BKT     0
4   1   1 17.0     LKT    45
5   2   1 17.3     LKT     0
6   3   1 17.2     LKT    17
7   1   1 17.0     RBT    14
8   2   1 17.3     RBT    13
9   3   1 17.2     RBT    17
```

The `dcast()` function is used to convert from long to wide format. As with the other functions mentioned so far, this function begins with the long format data frame to be reshaped. However, in contrast to all of the previous functions this function uses a formula to identify how the reshaping should occur. The left-hand-side (LHS; i.e., to the left of the tilde) of the formula contains the variables that represent or are constant for all individuals. The right-hand-side (RHS) of the formula contains the variable(s) that represent(s) the "groups" of each individual. Finally, the `value.var=` argument is used to identify the variable that contains the multiple measurments for each individual. Thus, the data shown above in long format is converted to wide format with

```
> ( dfW <- dcast(dfl,net+eff+temp~species,value.var="catch") )
  net eff temp BKT LKT RBT
1   1   1 17.0  17  45  14
2   2   1 17.3   5   0  13
3   3   1 17.2   0  17  17
```

# Reproducibility Information

## Version Information

- **Compiled Date:** Sat Mar 02 2013
- **Compiled Time:** 6:53:17 PM
- **Code Execution Time:** 4.92 s

## R Information

- **R Version:** R version 2.15.2 (2012-10-26)
- **System:** Windows, i386-w64-mingw32/i386 (32-bit)
- **Base Packages:** base, datasets, graphics, grDevices, methods, stats, tcltk, utils
- **Other Packages:** FSA_0.3.4, knitr_0.9, plyr_1.8, reshape_0.8.4, reshape2_1.2.2, rJava_0.9-4, RODBC_1.3-6, XLConnect_0.2-4, XLConnectJars_0.2-4
- **Loaded-Only Packages:** car_2.0-16, cluster_1.14.3, digest_0.6.3, evaluate_0.4.3, formatR_0.7, gdata_2.12.0, gplots_2.11.0, grid_2.15.2, gtools_2.7.0, Hmisc_3.10-1, lattice_0.20-13, multcomp_1.2-16, nlme_3.1-108, plotrix_3.4-6, quantreg_4.94, relax_1.3.12, sciplot_1.1-0, SparseM_0.96, stringr_0.6.2, TeachingDemos_2.9, tools_2.15.2
- **Required Packages:** FSA, RODBC, XLConnect, reshape2 and their dependencies (car, gdata, gplots, Hmisc, knitr, lattice, methods, multcomp, nlme, plotrix, plyr, quantreg, relax, reshape, rJava, sciplot, stats, stringr, tcltk, TeachingDemos, utils, XLConnectJars)