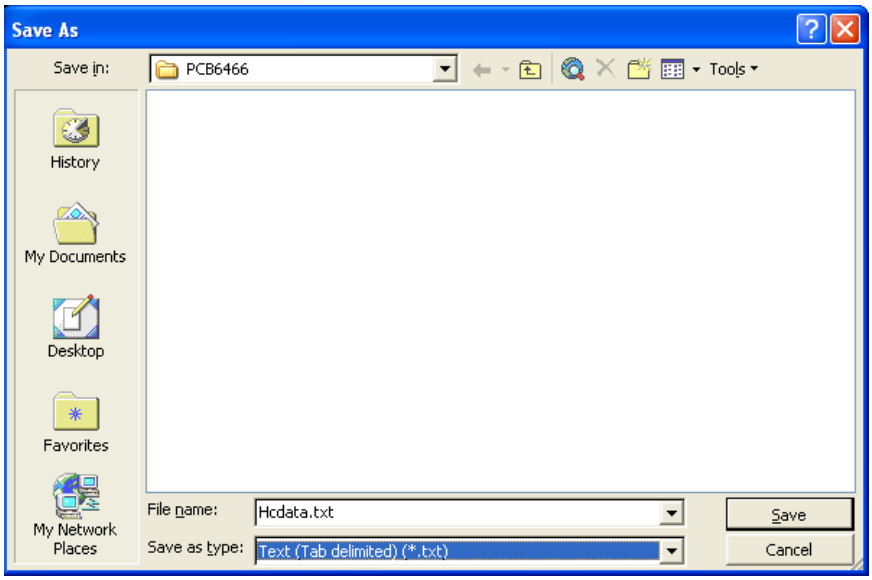
R Demonstration – Managing, checking and describing Data

Objective: The purpose of this week's session is to demonstrate how to convert data stored in Microsoft Excel format into a format that is usable in R. Also, basic R functionality for exploring and describing data will be introduced.

Part I. Exporting Data from Excel

In this part of the lesson, you will learn how to convert data stored in MS Excel into a tab-delimited data file that can be read into R. While the steps listed below are specific to the *Hypericum cumulicola* data file, you can follow the general process for your own data files with only a few minor modifications. Here are the steps:

1. Create a folder on your Desktop called `PCB6466`.
2. Download the *Hypericum* Excel spreadsheet (`Hcdata.xls`) from the course website and save it in your `PCB6466` folder.
3. Start MS Excel and open the spreadsheet you just saved. Notice that the first row of data is a series of column names and each of the other rows is a set of data for a single plant. The first column (tag) list the tag number that identifies each individual in the sample, the second to fourth columns the stage of the individual (0, dead; 1, alive; 3, new adult; 5, seedling; 9, previously dead), the fifth to the seventh columns (h94, h95, h96) list their heights in cm, and the last three list the number of reproductive structures per plant. If the plant died, NA (not available) was entered for height and number of reproductive structures.
4. Select *File→Save As…* from the Excel menu. When the "Save As" dialog box appears, choose **Text (Tab delimited)(*.txt)** as the "Save as type:" from the dropdown menu, and make sure the "Save in:" folder is the `PCB6466` folder you created in the first step. Finally, type `Hcdata.txt` for the file name and click the Save button. See the screenshot below:



Part II. Exploratory Data Analysis Using R

Reading the *Hypericum* data into R:
Start the R software. From the menu bar, select *File→Change dir…* and then browse to the `PCB6466` folder you created on the Desktop (NOTE: if you prefer doing this from the command line, you can use the `setwd(dir)` function instead).

Now enter the following at the command prompt:

```
> hc <- read.table("hcdata.txt", header=T)
```

This command uses the *`read.table`* function to create a variable named `hc`, which is a type of object in R known as a "data frame". Basically, a data frame is a tabular representation of data, with each row representing a single entity (e.g., a plant) and the columns representing the various data measurements for the entity. The first argument to the *`read.table`* function is the name of the tab-delimited text file from which to read the file. The `header=T` argument indicates that the first row in the data file is a list of column headings. There are many other options for using the *`read.table`* function, and you should consult the R help system for additional information.

Next, type the following commands at the prompt:

```
> attach(hc)
> names(hc)
```

The `attach(hc)` function instructs R to make the column headings available as variable names, and the `names(hc)` function displays these variable names. You should see the following output after you execute these two commands:

```
[1] "tag"  "SA94" "SA95" "SA96" "h94"  "h95"  "h96"  "R94"  "R95"  "R96"
```

As always, you can type the name of any of the listed variable names and R will display the contents of that variable.

Using the `tapply` function for exploratory data analysis:
In this section, we will use the *`tapply`* function to compute various summary statistics from our `hc` data frame. The description of the *`tapply`* function provided in the R help system may seem a bit intimidating, but we will be using a very narrow subset of its functionality. Generally, our usage of this function will follow this format:

```
> <output_var> <- tapply(<input_var>, <factor>, <stat>)
```

In this generalized form, `<input_var>` is the name of our input variable, `<factor>` is the name of the variable used to group the output data, and `<stat>` is the name of the summary statistic to be performed on the data. For example, to calculate the means of the height data from 1994 (`h94`) grouped by the status of the plant in 1994 (`SA94`), you would enter the following command:

```
> height.mean94 <- tapply(h94, SA94, mean, na.rm=T)
```

The final argument, `na.rm=T`, is an optional argument to the *`mean`* function indicating that all `NA` values (i.e., those for which no data are available) should be ignored when computing the mean. To see the results of this command, type the following:

```
> height.mean94
        3         5
34.486322  9.779944
```

These results show that two means were calculated for the plant heights in 1994: a mean of 34.49 for plants with a status of 3 (“new flowering plant”) and a mean of 9.78 for plants with a status of 5 (“new seedling”).

Similarly, we can use the *`tapply`* function to calculate and display the standard deviation (`sd`) and the number of observations (`length`) for the 1994 height data grouped by status with the following commands:

```
> height.sd94 <- tapply(h94,SA94,sd,na.rm=T)
> height.sd94
       3        5
12.870647  7.263134

> height.n94 <- tapply(h94,SA94,length)
> height.n94
  3   5
658 359
```

As a logical next step, you might wish to calculate these summary statistics for the 1995 plant height data. For example, you could get the mean as follows:

```
> height.mean95 <- tapply(h95,SA95,mean,na.rm=T)
> height.mean95
       0        1
     NaN 46.15604
```

Notice that for 1 of the statuses (0), the mean height is listed as `NaN`, which is an abbreviation for “Not a Number”. This occurs because all of the height values for these statuses are `NA` (i.e., no data), and you can’t take the mean of an empty set. Thus, we

need to use the *is.na* function to filter the height and status data from 1995 to exclude all the NA data, as follows:

```
> nh95 <- h95[!is.na(h95)]
> nSA95 <- SA95[!is.na(h95)]
```

The previous commands create 2 new variables, nh95 and nSA95, by using the !is.na function to filter out the NA values in the h95 variable. The ! in front of the is.na function tells R to only return those values that are not NA. It is then a simple matter to use the new variables nh95 and nSA95 to calculate the mean, standard deviation, and number of observations for the 1995 plant height data grouped by status:

```
> height.mean95 <- tapply(nh95,nSA95,mean)
> height.mean95
       1
46.15604

> height.sd95 <- tapply(nh95,nSA95,sd)
> height.sd95
       1
285.6694

> height.n95 <- tapply(nh95,nSA95,length)
> height.n95
  1
596
```

Creating scatterplots and boxplots:
An important component of exploratory data analysis is plotting your data to check for unusual values (known as "outliers") and/or data entry errors. We have altered two values in the sample for you to identify "twisted outliers". Scatterplots are often used to compare the relationship between two variables, while boxplots are useful for presenting the central value and spread of one or more variables. In this part of the lesson, we will use the R function *plot* to create scatterplots and *boxplot* to create boxplots. While we will only exercise the most basic options for these 2 functions, they both have many optional arguments that are described in the built-in R help system. Furthermore, the plot function can be used to create line graphs as well as scatterplots.

One important thing to note is that, when R starts, it defaults to a single window for plotting graphs. This means that, if you execute more than one plotting command in your session, the most recent graph will overwrite any previously drawn graph. To get around this limitation, we will use the *par* function to create a graphical display with multiple plots (NOTE: the *par* function can be used to set many other graphical parameters in R. Once again, check the documentation for more details). Enter the following command:

```
> par(mfrow=c(1,3))
```

The `mfrow=c(1,3)` argument indicates that the display should be divided into 1 row and 3 columns. Now type the following series of `plot` commands:

```
> plot(h94,R94)
> plot(h95,R95)
> plot(h96,R96)
```

The first `plot` command creates a scatterplot of height in 1994 (`h94`) on the X axis versus reproductive structures produced in 1994 (`R94`) on the Y axis, and the other 2 commands create similar scatterplots based on the 1995 and 1996 data, respectively.

The following series of commands will create a set of boxplots of the height data in 1994, 1995, and 1996:

```
> par(mfrow=c(1,3))
> boxplot(h94,na.rm=T)
> boxplot(h95,na.rm=T)
> boxplot(h96,na.rm=T)
```

Here again, the `na.rm=T` is an optional argument that tells the `boxplot` function to ignore any `NA` values in the data. Finally, we use the `par` function to reset the graphical display to its original setting:

```
> par(mfrow=c(1,1))
```

Correcting erroneous data:
From the boxplot of the 1995 height data, it should be quite apparent that there is an outlier in the data. Often, you will not know whether such an outlier reflects a real biological process or whether it represents an error during the data collection or the data entering (see pages 212-216 of Gotelli & Ellison 2004 for a discussion of outliers vs. errors).

For this exercise, however, we will assume that you know that the extreme value is an error and that the correct value should be 32. A basic approach to fixing this error in R is to find the row index for this extreme value in our `h95` variable, and then to update this index with the correct value. First, we use the *`max`* function to determine the actual value of this outlier and store this value in a variable named `max.h95`:

```
> max.h95 <- max(h95,na.rm=T)
```

Next, we use the following commands to create a set of indices for the `h95` variable and then associate them with that variable:

```
> index <- 1:length(h95)
```

```
> subhc <- cbind(index,h95)
```

The first command creates the indices and stores them in the vector named `index`, and then the `cbind` function joins the `index` and `h95` variables together into a new data frame named `subhc`. Finally, we use the following pair of commands to capture the proper index to update in the variable `index.h95` and then we actually update the h95 data at that index with the corrected value of 32:

```
> index.h95 <- index[h95==max.h95 & !is.na(h95)]
> h95[index.h95] <- 32
```

You can check that the outlier has been corrected by drawing another boxplot:

```
> boxplot(h95,na.rm=T)
```

Part III. Exercise 2

Please perform the following exercises to practice the R functions learned in this demo and explore other functions. **Submit a single Word document with your answers and the R code you used to obtain your answers. DUE: 09/10/2013 (2 points each).**

1. Using the approach demonstrated in the "Correcting erroneous data:" section of Part II, correct any other data errors you can detect in the data of the previous exercise.
2. Calculate the means of the height data from 1995 after eliminating the "twisted outliers". Repeat all the plots for the corrected data.
3. Calculate the medians of the height data from 1994 (`h94`) grouped by the status of the plant in 1994 (`SA94`). You may wish use the functions *`tapply, median,`* and *`summary`*.
4. Use the *`table`* function to create contingency tables of the stages of plants in 1994 versus 1995 and in 1995 versus 1996.
5. Examine the tables you created in the exercise above to detect any logical errors in the stage data. For example, a plant classified as "dead" in 1994 that is then classified as a "new flowering plant" in 1995 is clearly an error. You may wish to use the *`factor`* function to assist you in detecting these errors.