How to import and merge many Excel files; each with multiple sheets of data...for statistical analysis.

As published in Benchmarks RSS Matters, August 2013

http://web3.unt.edu/benchmarks/issues/2013/08/rss-matters

Jon Starkweather, PhD

Jon Starkweather, PhD
jonathan.starkweather@unt.edu
Consultant
Research and **S**tatistical **S**upport

http://www.unt.edu

http://www.unt.edu/rss

RSS hosts a number of "Short Courses".
A list of them is available at:
http://www.unt.edu/rss/Instructional.htm

Those interested in learning more about R, or how to use it, can find information here:
http://www.unt.edu/rss/class/Jon/R_SC

How to import and merge many Excel files; each with multiple sheets of data...for statistical analysis.

This month's article was motivated by the need to import and merge together multiple Excel files and the multiple sheets within each Excel file. Excel is extremely popular as a tool for organizing data and it has fairly easy-to-use functions for rudimentary statistics and data displays (i.e. graphs & charts). However, it is not a statistical software package and therefore, it is often necessary to import Excel data structures into other, more statistically oriented software. For this reason, RSS personnel do not recommend using Excel; for data storage, data display, or data analysis. An often quoted phrase[1] is the following; the only thing worse than using SPSS, is using Excel. For more information on the known problems with Excel and other spread sheet based software, see Burns (2013). RSS recommends storing data in plain text (.txt) files with comma delimiters; also known as a comma separated values (.csv) file type. The reason RSS recommends text (.txt) or comma separated values (.csv) file types is because those file types can be easily opened or imported into all the statistical software packages. However, if you feel you must use Excel, then this article should help you with the inevitable task of getting data from Excel into a more worthy software package for statistical data analysis; and there really is no more worthy software for that purpose than **R**[2].

Context of the Example

An example has been created to illustrate a procedure for importing several Excel files, each with multiple sheets, into the R workspace and merging them together as a single data frame. The premise of our example is a research design with 10 participants, 3 lighting conditions, and 5 time series (chin movements, left eye [pupil] movements, right eye [pupil] movements, left wrist movements, right wrist movements). Each participant was exposed to each lighting condition and their movements were measured throughout a 10 minute typing task – all three body-part measuring apparatus' took samples 100 times per minute to measure positional changes (in millimeters) from an enforced baseline / start position, while each eye's pupil movement reflects the movement (in millimeters distance) from looking at the center of the screen. In other words, the eye (pupil) movement refers to changes of movement in gazing at the center of the screen to gazing at the edges of the screen, or the keyboard. Again, the time series data was sampled at 100 times per minute for the full 10 minutes of typing (n = 1000, per time series). Motion capture software exported the resulting data into 10 Excel files. Each Excel file corresponds to each participant (participant.1.xls, participant.2.xls...etc.) and each Excel file contains 3 sheets; one sheet per lighting condition (Off, Dim, Bright). Each sheet contains five time series corresponding to the five measured variables (Chin, R_eye, L_eye, R_wrist, L_wrist). The resulting simulated data is available on the RSS servers so that the reader can download the data files[3] and replicate what is illustrated below. Our goal was to import all the data and merge it into a single data frame.

[1]The phrase is believed to have originated with respected statistician and prominent R user Frank Harrell of Vanderbilt University at the 5th annual Bayesian Biostatistics Conference.

[2]http://cran.r-project.org/

[3]The data can be downloaded from the following links:

```
http://www.unt.edu/rss/class/Jon/Benchmarks/ExcelFiles/participant.1.xls
http://www.unt.edu/rss/class/Jon/Benchmarks/ExcelFiles/participant.2.xls
http://www.unt.edu/rss/class/Jon/Benchmarks/ExcelFiles/participant.3.xls
```

Illustrative Example

First, 'set' the working directory (wd) to the (path) location on your computer where the files are located; in this example, we have the 10 Excel files on our desktop. Below, and throughout the example, we are using black, Times New Roman, font for text and we are using Courier New font for R script (in red) and R output (in blue).

```
setwd("C:/Users/jds0282/Desktop/")
```

Next, load the packages which will allow us to import Excel data files; the XLConnect package is the package we want and it requires the rJava package.

```
library(rJava)
library(XLConnect, pos = 4)
XLConnect 0.2-5 by Mirai Solutions GmbH
http://www.mirai-solutions.com ,
http://miraisolutions.wordpress.com
```

Next, create an object with the file names. Here, we are using the paste function to create sequential character string names.

```
pre1 <- "participant"
pre2 <- seq(1:10)
suf <- "xls"
file.names <- paste(pre1, paste(pre2, suf, sep = "."), sep = ".")
rm(pre1, pre2, suf)
file.names
[1] "participant.1.xls"  "participant.2.xls"  "participant.3.xls"
[2] "participant.4.xls"  "participant.5.xls"  "participant.6.xls"
[3] "participant.7.xls"  "participant.8.xls"  "participant.9.xls"
[4] "participant.10.xls"
```

Next, create an object with the sheet names. Recall, each file contains 3 sheets; each sheet corresponds to a lighting condition.

```
sheet.names <- c("Off","Dim","Bright")
sheet.names
[1] "Off"    "Dim"    "Bright"
```

Next, we create a vector of names which will be the column names for the final data frame. The data

```
http://www.unt.edu/rss/class/Jon/Benchmarks/ExcelFiles/participant.4.xls
http://www.unt.edu/rss/class/Jon/Benchmarks/ExcelFiles/participant.5.xls
http://www.unt.edu/rss/class/Jon/Benchmarks/ExcelFiles/participant.6.xls
http://www.unt.edu/rss/class/Jon/Benchmarks/ExcelFiles/participant.7.xls
http://www.unt.edu/rss/class/Jon/Benchmarks/ExcelFiles/participant.8.xls
http://www.unt.edu/rss/class/Jon/Benchmarks/ExcelFiles/participant.9.xls
http://www.unt.edu/rss/class/Jon/Benchmarks/ExcelFiles/participant.10.xls
```

frame must include columns (factor level variables) which contain coding information which identifies each row's data. In this example, we need three such factors; one for the participant, one of the condition, and one for the sampling frame (1 to 1000) which represents each of 100 samples per minute (for 10 minutes). The other five names (and columns) represent the five motion capture time series distance measures.

```
e.names <- c("participant.id","condition","sampling.frame",
+              "Chin","R_eye","L_eye","R_wrist","L_wrist")
e.names
[1] "participant.id" "condition"      "sampling.frame" "Chin"
[2] "R_eye"          "L_eye"          "R_wrist"        "L_wrist"
```

The last step in preparation is to create the final data frame (data.1), keep in mind, this data frame only has one row (for now) and that row includes only 'NA' values. However, some simple mathematics allows us to compute the size of the final data frame. It will have 8 columns and 30,000 rows (10 participants * 3 conditions each * 1000 rows per condition). It is important to remember the first row is made up of 'NA' values and represents a place holder (it will be deleted after all the data is imported).

```
data.1 <- data.frame(matrix(rep(NA,length(e.names)),
     ncol = length(e.names)))
names(data.1) <- e.names
data.1
participant.id condition sampling.frame Chin R_eye L_eye R_wrist L_wrist
1             NA        NA             NA   NA    NA    NA      NA      NA
```

Now, we're ready to use two 'for-loops' to import each sheet of each file and row bind (rbind) them to the original / final data frame. However, it may be beneficial to elaborate on what each line of each 'for-loop' is doing. Line numbers have been added to the script below in order to help facilitate explanation of each line. Obviously, these line numbers are not functional R script (red, Courier New) or R output (blue, Courier New) and therefore are printed in black (Times New Roman) font.

```
1:  for (i in 1:length(file.names)){
2:    wb <- loadWorkbook(file.names[i])
3:    for (j in 1:length(sheet.names)){
4:      ss <- readWorksheet(wb, sheet.names[j], startCol = 2, header = TRUE)
5:      condition <- rep(sheet.names[j],nrow(ss))
6:      sub.id <- rep(file.names[i],nrow(ss))
7:      s.frame <- seq(1:nrow(ss))
8:      df.1 <- data.frame(sub.id,condition,s.frame,ss)
9:      names(df.1) <- e.names
10:     data.1 <- rbind(data.1, df.1)
11:     rm(ss, condition, s.frame, sub.id, df.1)
12:     }
13:   rm(wb)
14:   }; rm(e.names, file.names, i, j, sheet.names)
```

Line 1 above simply initiates a ‘for-loop’; which is nothing more than a way to tell the computer to read all the lines between the curly braces and before proceeding, it should read those lines again, and again, and again...until ‘i’ equals the length of the ‘file.names’ object. The length of the ‘file.names’ object is 10 because we specified earlier 10 file names. So, line 1 is essentially instructions which say; read the following lines, or iterate through the following lines, 10 times. The character ‘i’ is assigned a zero until the first iteration is complete, at which time it is assigned a 1; next iteration i = 2, and so on until i = 10. The closing curly brace is on line 14 and the script after that curly brace will only be read when all 10 iterations have completed. So, lines 2 through 13 will each be read, or processed, 10 times in sequence (i.e. read lines 2 through 13, then read lines 2 through 13, then...etc.).

Line 2 above simply imports an Excel workbook (file) and assigns it to ‘wb’ (an arbitrary or temporary name of the workbook). We are telling the software the file name to look for by passing the file.names object to the loadWorkbook function and because the file.names object contains all 10 names, we specify the one which corresponds to the iteration number (i). So, for the first iteration, the loadWorkbook function looks for “participant.1.xls” because that is the first object of the file.names object.

Line 3 initiates a second ‘for-loop’ but instead of labeling each iteration ‘i’ we are labeling each iteration in this loop ‘j’ - which differentiates the iterations of the two loops. The ‘j’ loop will iterate from 1 until the length of the sheet.names object. Recall, we specified 3 sheet names; corresponding to the 3 lighting conditions (Off, Dim, Bright). Keep in mind, the closing curly brace for the ‘j’ loop is on line 12; which means, there will be 3 iterations of loop ‘j’ occurring inside each single iteration of the ‘i’ loop. Another way to think about this is; we read in an Excel file with the ‘i’ loop and that file contains 3 sheets which must be imported before going to the next Excel file.

Line 4 imports or reads the j^{th} sheet and assigns it as an object of ‘ss’. The ‘ss’ is simply an arbitrary or temporary name for the sheet. Each sheet contains the data from the five measurements (chin, right eye, left eye, right wrist, left wrist) – this includes 1000 time series data points for each of the five measures or columns. Take note of the arguments of the readWorksheet function. First, we pass the wb object (the workbook) to the readWorksheet function, then we specify which sheet to import using the vector of sheet names (here, the j^{th} sheet, with j = to the iteration number of the ‘j’ loop). Subsequent arguments allow us to specify the particular column and row (startCol; startRow; Header = TRUE or FALSE) of the sheet which contains the data. We could (although not shown) use other arguments (endCol; endRow) to specify specific places in the sheet to stop reading or importing data.

Line 5 simply creates a vector containing the sheet name (of the sheet just imported) replicated the same number of times as the number of rows of that sheet (n = 1000) and assigns that vector the name ‘condition’. Line 6 does the same thing for the workbook name or Excel file name which corresponds to the participant whose data is being imported. Line 7 creates a vector of sequential values from 1 to the number of rows of the sheet being imported. These values simply number each sample from the motion capture software (1000 samples = 100 samples per minute of the 10 minute task). Line 8 simply creates a temporary data frame (df.1) which has 1000 rows and 8 columns. The columns correspond to the participant identification (participant.id), the sheet name or condition (1 of three lighting conditions), the sequential sampling frame numbers (1 to 1000) and then the five motion capture measures (chin, right eye, left eye, right wrist, left wrist). Line 9 assigns the proper names to these columns, which are the same names and will match the columns of the final data frame (data.1). Line 10 ‘row binds’ (rbind) the newly imported data (df.1) to the bottom of the final data frame (data.1) - simply adding rows to the final data frame.

Line 11 removes (rm) all the no longer needed objects. Line 12 ends the ‘j’ loop. Line 13 removes (rm) the no longer needed workbook (wb). And finally, line 14 ends the ‘i’ loop and then removes objects

no longer needed. Line 11 and line 13 are not strictly necessary because each iteration of each loop will re-write the objects contained in those lines. However, programming has some best practices which can be described as similar to some rules learned in kindergarten...always share and always cleanup after yourself.

Now, to point out one of the benefits of using R: after having read the above section and having studied the R script it describes; it is plain to see that an object oriented programming language, such as the R programming language, is much more efficient than written American English. It took several paragraphs to explain only 14 lines of programming.

Once the looping functions have completed (it should take less than 10 seconds), you can run a summary of the final data frame. You'll notice there are some oddities associated with the data frame, which are revealed in the summary output.

```
summary(data.1)
participant.id      condition          sampling.frame        Chin
Length:30001       Length:30001       Min.   :   1.0   Min.   :-501.606
Class :character   Class :character   1st Qu.: 250.8   1st Qu.:-249.776
Mode  :character   Mode  :character   Median : 500.5   Median :   0.044
                                      Mean   : 500.5   Mean   :  -1.056
                                      3rd Qu.: 750.2   3rd Qu.: 247.143
                                      Max.   :1000.0   Max.   : 501.578
                                      NA's   :1        NA's   :1
    R_eye               L_eye              R_wrist             L_wrist
Min.   :-5.022   Min.   :-5.018   Min.   :-502.524   Min.   :-502.926
1st Qu.:-2.504   1st Qu.:-2.508   1st Qu.:-249.220   1st Qu.:-249.948
Median : 0.000   Median :-0.001   Median :  -0.330   Median :   0.234
Mean   : 0.008   Mean   :-0.000   Mean   :  -0.994   Mean   :  -0.718
3rd Qu.: 2.500   3rd Qu.: 2.521   3rd Qu.: 248.641   3rd Qu.: 248.372
Max.   : 5.013   Max.   : 5.028   Max.   : 503.390   Max.   : 502.568
NA's   :1        NA's   :1        NA's   :1           NA's   :1
```

The first thing to notice is the participant identification (participant.id) and condition columns contain character string information instead of factor level data. Also, notice the number of rows (for all columns) is 30001 instead of 30000. The extra row is the first row of the data frame which contains all NA as a result of how we created the data frame prior to importing the data. So, we need to remove the first row and we need to convert the first two columns to factors.

```
data.1 <- data.1[-1,]
data.1[,1] <- factor(data.1[,1])
data.1[,2] <- factor(data.1[,2])
summary(data.1)
           participant.id   condition     sampling.frame        Chin
participant.1.xls : 3000   Bright:10000   Min.   :   1.0   Min.   :-501.606
participant.10.xls: 3000   Dim   :10000   1st Qu.: 250.8   1st Qu.:-249.776
participant.2.xls : 3000   Off   :10000   Median : 500.5   Median :   0.044
participant.3.xls : 3000                  Mean   : 500.5   Mean   :  -1.056
participant.4.xls : 3000                  3rd Qu.: 750.2   3rd Qu.: 247.143
```

```
participant.5.xls : 3000                  Max.   :1000.0   Max.   : 501.578
(Other)           :12000
    R_eye               L_eye              R_wrist             L_wrist
Min.   :-5.022   Min.   :-5.018   Min.   :-502.524   Min.   :-502.926
1st Qu.:-2.504   1st Qu.:-2.508   1st Qu.:-249.220   1st Qu.:-249.948
Median : 0.000   Median :-0.001   Median :  -0.330   Median :   0.234
Mean   : 0.008   Mean   :-0.000   Mean   :  -0.994   Mean   :  -0.718
3rd Qu.: 2.500   3rd Qu.: 2.521   3rd Qu.: 248.641   3rd Qu.: 248.372
Max.   : 5.013   Max.   : 5.028   Max.   : 503.390   Max.   : 502.568
```

Now that we have the data imported and merged into a single data frame, we can then export that data frame by writing it to our working director, which was set at the beginning of the script ('setwd') to our desktop. The file which is saved to the desktop will be named "typing_experiment_data.txt" and it will contain comma delimited (or comma separated) values, without row names but with column names. Any missing data (there is none in this example) will be recognized as 'NA' and a decimals will be represented with a period ('.').

```
write.table(data.1, file = "typing_experiment_data.txt",
   sep = ",", na = "NA", dec = ".", row.names = FALSE,
   col.names = TRUE)
```

Conclusions

Keep in mind, there are a variety of different ways of accomplishing what was accomplished in this article. The example here merged all the data into one data frame. Different situational needs might dictate keeping the data separated by participant (i.e. workbook or file) or separated by condition (i.e. sheet); in those instances it may be preferable to import the data structures to multiple list objects or multiple data frames. That is another benefit of using R, the flexibility it affords the analyst in deciding what to do and how to do it. An R script[4] file with the same information as contained in this article is available at the Research and Statistical Support Do-It-Yourself Introduction to R course website[5]. Lastly, for those interested in seeing how the example data was created in R, and how it was exported from R into Excel.xls files; please take a look at the script[6] which was used.

Until next time, *you can't always get what you want, but if you try sometimes...*

[4] http://www.unt.edu/rss/class/Jon/R_SC/Module3/ImportManyExcel.R
[5] http://www.unt.edu/rss/class/Jon/R_SC/
[6] http://www.unt.edu/rss/class/Jon/R_SC/Module3/MultiExcelDataCreation.R

References & Resources

Burns, P. (2013). Spreadsheet Addiction. Available at:
http://www.burns-stat.com/documents/tutorials/spreadsheet-addiction/

This article was last updated on August 21, 2013.

This document was created using LaTeX