The **DSA** *Do It Yourself* Introduction to **R** Short Course Manual

Jon Starkweather, PhD

November 28, 2018

Jon Starkweather, PhD
`jonathan.starkweather@unt.edu`
Consultant
Data **S**cience and **A**nalytics

`http://www.unt.edu`

`http://it.unt.edu/research`

DSA hosts a number of "Short Courses".
A list of them is available at:
`http://it.unt.edu/researchshortcourses`

The material contained in this manual can also be found at:
`http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/R_SC/`

Contents

1 Module 1: Download and Install R

1.1 Introductory Notes 1

1.1.1 Downloading R.

When you first go to CRAN[1] to download R, you will be prompted to select which operating system you will be using. Once you click on Windows you will be confronted with two choices; base and contrib. You only need the base. Later we will install packages which allow you enjoy all the functioning of R. Once you click on base you will be confronted with a page showing (in bold) a link to Download R x.xx.x for Windows where the series of x indicate the current version (e.g. 3.3.1). Once you click on the download link; you will be prompted to save the file somewhere on your computer. Saving it to the desktop is fine; there will be no need to keep it after you have installed R.

1.1.2 Installing R.

Double click on the executable file to install R. The default options/settings as specified during installation will be fine. The program can very easily be customized after installation.

There are three windows youll likely use every time you use R. The GUI or console window is the core of the program and can act as both textual input and textual output display. The graphics window, which as the name implies, displays graphical output (e.g. histograms, scatter plots, topographical displays of terrain, 3D perspective plots, thermal images, etc.). The script window displays script (also called syntax, or program code, or input code), which is not necessary, but often preferred as a way of building script with comments (i.e. not working code but disregarded by the console) and proof reading it prior to submitting it for processing...and saving it for later. Once you complete the first three modules of this tutorial website, I would strongly encourage installing and using RStudio – it is free and has quickly become the most popular way of using R. More information on RStudio can be found at the RStudio home page[2]. RStudio has a panel or 'pane' layout which can be changed by going to "Tools" — "Global Options" — then "Pane Layout" so you can arrange the panes however you want. You can also minimize a pane or panes; which is common, so that only the "Source" or scripting pane and the console pane are shown. RStudio also has a default hotkey for submitting highlighted script; pressing both the "Ctrl" and "Enter" keys will submit highlighted script to the console and automatically return focus back to the scripting or "Source" pane. Also, graphs or plots can be brought into a separate window outside of the RStudio suit by clicking on the "Expand" button of the "Plots" pane – often helpful to enlarge a plot to evaluate its contents. A brief introduction to RStudio[3] may help users become familiar with some of the settings and some preferences using R through RStudio.

[1] `http://cran.r-project.org/`
[2] `https://www.rstudio.com/`
[3] `http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/R_SC/Module1/RStudio_Intro.pdf`

1.1.3 The first time you open R.

You will be confronted with a window in a window. It is recommended (but not necessary) you change the display by changing the GUI preferences; GUI stands for Graphical User Interface. At the top of the window, click on Edit then click on GUI preferences. Many of these preferences are self explanatory; but here are a few I use: SDI and single window which changes the console/GUI to a single window display once applied and changes are saved. I prefer font size of 12 which is a little larger and easier for my old eyes to read, but not so large as to dramatically cut down the amount of character lines displayed. I generally change the Pager rows and columns so that the console window is quite largethe specific numbers will be dependent on your screen size/resolution; so it may take some trial and error fiddling to get what you want. I also set the Initial left and top to zero so that the console window opens at the top/left of my screen when I open R. Again, these are merely preferences and each of you should set the program up in a way which is most comfortable for you. Once you have the preferences set how you want them, you will need to SAVE them; regardless of whether or not you apply them. To save your GUI preferences, click the Save button on the bottom of the GUI preferences. You will be prompted to select a directory in which to save the Rconsole. You must save this file in the etc directory if you want the changes to be present each time you open the R program; this directory is located inside the R directory where you installed the program, generally at:

File paths are listed in green

```
C:\Program Files\R\R-x.xx.x\etc
```

where x.xx.x refers to the version number of the R installation you have (e.g. R-3.3.1). Once you have saved the GUI preferences, close the program (no need to save workspace image; more on this later) and open it again to make sure the changes have been saved and are being applied.

Congratulations; you now have a working, albeit limited, version of R – and it was completely FREE!

1.2 Introductory Notes 2

1.2.1 Some initial orientation to using R.

There are some key terms you will need to become familiar with; first R is an object oriented system. Anything can be an object; a score, a series of scores (also called a vector), a named variable (also called a vector), a matrix (made up of rows and columns), a data frame, or a list (a larger group of objects). As an example; lets open R and create a new object called ‘x’. Our object x will initially be something as simple as an individual score, say 5. To communicate this in R, in the console type the following and hit the enter key at the end of each line:

R script is listed in red

```
x <- 5
x = 5
x<-5
x=5
x
[1] 5
```

R output is listed in blue

Notice how both = and `<-` refer to the same operation and the presence of spaces before and after are not necessary. The operation = and `<-` perform is assignment; in other words, we have assigned

5 to x. Now we can perform simple arithmetic or complex algebra using our assigned value for x. For example, try any of the following:

```
x - 5
[1] 0
x / 5
[1] 1
x ^ 2
[1] 25
x ^ 1/2
[1] 2.5
x * 6
[1] 30
x6
Error: object "x6" not found
6x
Error: unexpected symbol in "6x"
```

Notice above that x * 6 is functional, while 6x and x6 are not.

```
x^2 + 5 * x - (x)
[1] 45
5 * x * (x^2)
[1] 625
5 * (x * x)^2
[1] 3125
```

We can also assign a group of values to our object using the concatenate or combine function which as a default produces a vector. We can then assign our vector (x) to another object or use it in a more complex function.

```
x <- c(5, 6, 7, 9, 10, 4, 5)
x
[1]  5  6  7  9 10  4  5
x + 2
[1]  7  8  9 11 12  6  7
is.vector(x)
[1] TRUE
y <- x * 2
y
[1] 10 12 14 18 20  8 10
```

You can continue to explore the arithmetic functions of R, but Im sure you didnt come to learn how R can be used as a calculator, so lets continue to the next module.

* Nifty trick in R console type: demo(graphics) then continue slowly hitting the enter key until nothing happens.

2 Module 2: Packages and Libraries

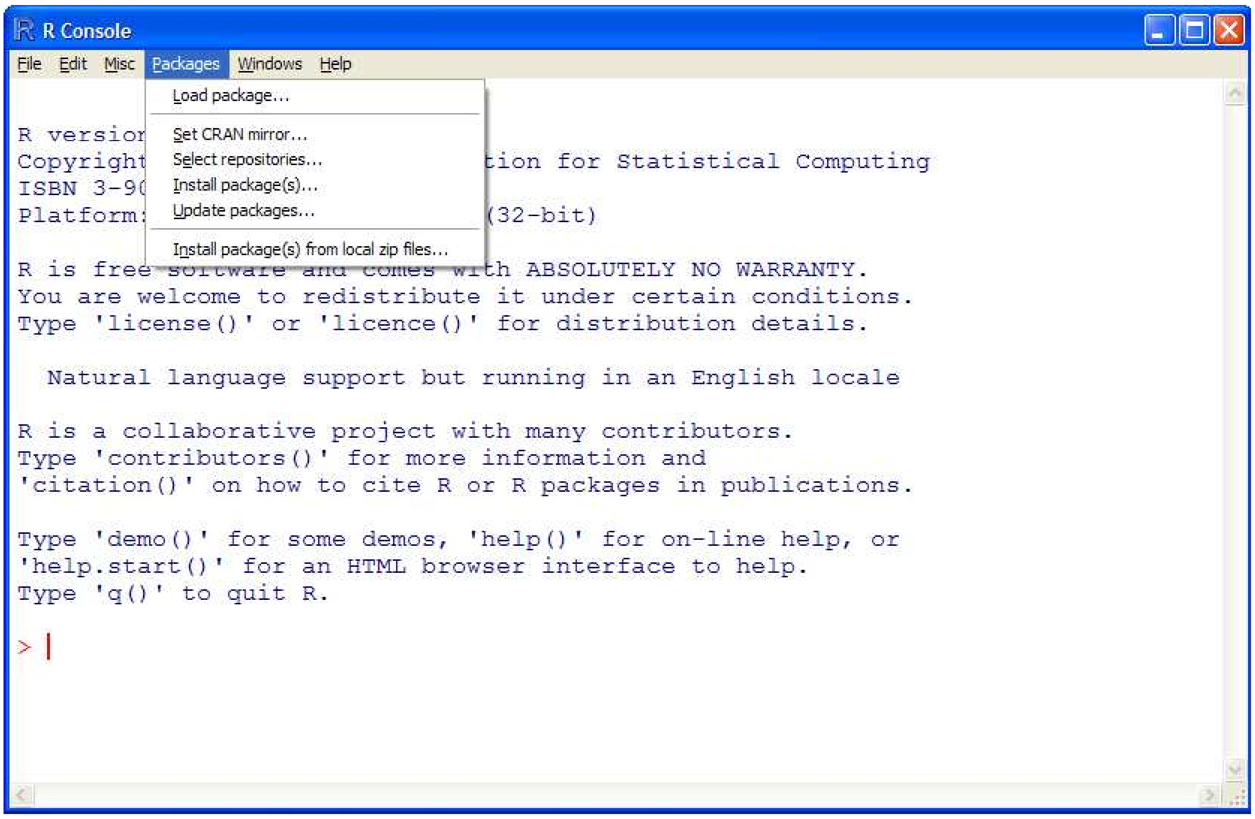
2.1 Packages

2.1.1 Explanatory Notes.

The terms package and library tend to be used interchangeably in R literature. These terms refer to the compiled chunks of downloadable content that developers and users create to increase the functionality of R. These packages are what make R so attractive and so capable. As an example, consider a fictional researcher, Dr. Smarty Pants at the University of Jupiter's Moon. Dr. Smarty Pants wants to do a new statistical technique, called the Wiz-Magic Decomposition analysis or WMD for short. Unfortunately, because WMD is so new, Dr. Smarty Pants can not find WMD in any of the existing statistical software available. But, Dr. Smarty Pants is an R user. So, no matter who Dr. Smarty Pants happens to be, where he or she happens to be, and no matter what analysis he or she wants to perform; any individual, like Dr. Smarty Pants, can write the code to perform the desired analysis and send it to CRAN as a new package. CRAN will then check it to make sure it works, has proper documentation, and post it so that everyone can then use the newest most advanced techniques, like WMD. You might think this process takes a great deal of time, but it does not. As of this writing, according to CRAN there are a over 9000 packages available and it is very likely that within a week, new packages will be available. Remember, packages are not just new analysis; many are very specific and may include better ways to do existing functions (e.g. the AMORE package is described as "A MORE flexible neural network package"). Furthermore, packages get updated to increase functionality or ease of use. Keep in mind, all packages and all new versions of R are completely free. So, you're now likely wondering, how do I get and use these packages? First, open R if it is not already.

2.1.2 Download and Install Packages.

To download and install packages, you must have R open and you must have an Internet connection. Next, click on 'Packages' at the top of the R Console.



```
R versio
Copyright                              tion for Statistical Computing
ISBN 3-9
Platform:                              (32-bit)

R is free                        with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

Take note of the options here, you will likely use two of them most frequently; 'Install Package(s)...' and 'Update packages...' The base install of R comes with about 5 core packages. We are interested in installing new/different packages; so, click on 'Install Package(s)...' You will then be prompted to select a CRAN mirror site from which to download packages. I suggest selecting a location close to the physical location of your computer. Once you select a mirror site, you will be presented with an alphabetical list of all the available packages. **Before you choose**, please take a minute to read the following paragraph.

The first time you install R on your machine, it is recommended you install all the packages used on this site. Fortunately, you do not need to point and click each of these packages to install them. You can use this[4] script. If you simply open that script in your browser and copy – paste its contents into your R console and run it, it will install all the package used on this site (and their dependent packages). Depending on your internet connection speed, this operation may take several minutes.

2.1.3 Choose Packages.

Now choose the Hmisc package; then click 'OK'. You will notice in the R Console, packages will be downloaded first, in the appropriate location, and then they will be installedwhich gives the message "package 'Hmisc' successfully unpacked and MD5 sums checked". You will also notice a message telling you where the temporary file is located which contains the downloaded package(s). You could delete this temp file after all the packages are installed, but it can be useful

[4]`http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/R_SC/Module2/InstallPackages_lis`

to take note of it in the unlikely event your Internet connection gets interrupted during download. If your connection is interrupted, you can still install the packages that were downloaded successfully by clicking on ‘Packages’ then ‘Install package(s) from local zip files...’.

2.1.4 Updating packages.

Now that you have the Hmisc package installed, you need to update. This may seem silly because you just installed it, but remember with a few thousand packages it becomes time consuming to update all of them when a new version of R is released and you may not have the most recent version of a package. So, click on ‘Packages’ and then ‘Update packages...’. If you have not done so already in this session of R (e.g. if you closed R and just re-opened it), you will need to choose a mirror site. You will then be prompted with a list of packages that can be updated. I generally choose all the available packages to make sure I have the most functionally up-to-date software. Once you choose which/all packages, click ‘Ok’ and you will notice a similar series of messages in the console showing the download and install of the updated packages.

We will be using the Hmisc package/library in the next set of notes on using a package/library; but in future notes/tutorials we will need a variety of packageagain, it is highly recommended you download and install all of the available packages.

2.2 Libraries

2.2.1 Loading a Library.

Loading a library is necessary to use the downloaded and installed package. Remember, the terms package and library are used synonymously throughout R literature; however, one could say a package becomes a library when it is loaded. In the previous set of notes, we downloaded and installed the Hmisc package (hopefully you have all packages used on this site now). To use a package, we must now load it. This is a very simple procedure; in the console, simply type the following and hit enter.

```
library(Hmisc)
Attaching package: 'Hmisc'
The following object(s) are masked from 'package:car':
recode
```

Any downloaded and installed package can be loaded by typing: library(name) where ‘name’ refers to the package name. Occasionally, when a package loads, it will ‘mask’ objects in other packages and the R console will return a message for each library loaded. As an example, consider the following; package Hmisc will mask the object ‘recode’ in package car if car is already loaded. This bears watching because; it can be extremely frustrating when attempting to use a familiar function which will no longer work because a newly loaded package is masking it. The good news is that this does not occur frequently even when using many libraries (e.g. multiple libraries can be loaded and used simultaneously). For this reason, it can be important and preferable to clean up after one’s self by using the ‘detach’ library command for libraries which are not being used.

2.2.2 Detaching a Library

Detach a library. Again, it is extremely easy to detach a library which is no longer needed. Simply type the following and hit enter:

```
detach("package:Hmisc")
```

Common errors result from forgetting the parentheses and/or quotations, as well as forgetting the colon between package and the library name. As an exercise in ensuring we are doing what we think we are doing, type the following and hit enter:

```
search()
[1] ".GlobalEnv"        "package:stats"     "package:graphics"
[4] "package:grDevices" "package:utils"     "package:datasets"
[7] "package:methods"   "Autoloads"         "package:base"
```

The search function shows all the loaded libraries. Notice, Hmisc is not listed. Now hit your up arrow 3 times and then hit the enter key. Notice, the up arrow scrolls through your previous commands and the third command up should have been ‘library(Hmisc)’. When you hit the enter key, you should have loaded that library again. So, if we hit the up arrow twice, we should see the ‘search()’ command and if we hit enter again, we should see that Hmisc is listed as a loaded package. Also notice that the ‘base’ package is loaded; which as the name implies is part of the base install of R and is loaded when the program R is opened/started.

2.2.3 Preloading Libraries

It can be very convenient to have one or a few libraries load when you start R if you tend to use these libraries during every session. This is the same rationale behind the base library being loaded each time R starts; everyone uses it every time they use R. You can customize R to load certain libraries upon start up by locating a file called “Rprofile.site” which by default is located:

```
C:\Program Files\R\R-x.xx.x\etc
```

where x.xx.x refers to the version number of the R installation you are using (e.g. R-3.3.1) and yes, there is an ‘etc’ folder. Once you locate the Rprofile.site file, you can open it in Notepad which by default is available with each installation of Windows. Here is where you can tell R to load certain libraries upon start up. As an example, you can see my current Rprofile.site file here[5]. One thing to notice in my Rprofile is the line:

```
setwd("C:\\Users\\jon\\Desktop\\Work_Stuff\\Jon_R")
```

which sets the working directory for R (Windows XP machine). This tells R where to start looking when ever you go to File and open or save in R. In other words, if you wanted to open or save something from the R console, it will start with the specified working directory. You can set your working directory to be any folder on your computer.

Important Note: If you change your Rprofile to preload libraries at start up, it will be necessary to cut and paste the Rprofile file from the ‘etc’ folder to some other location prior to starting R when you want to update packages. For instance, after having R and all packages downloaded and installed for a week, you will likely want to check for updates. Then, go to the ‘etc’ folder and cut

[5]http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/R_SC/Module2/JonsRprofileFile.tx

the Rprofile file out and paste it to your desktop. Then, start R and update packages/libraries as mentioned previously. Yes, the program will function without that file. After updating all packages, then close R. Then, cut the Rprofile file off your desktop and paste it back into the 'etc' folder...then restart R. The reason you would need to do this is because, R will not update packages/libraries that are in use (i.e. loaded). So, if you have your Rprofile set to load some libraries at start up, you would never be able to update those libraries.

So, you're probably wondering what's in a library? We'll answer that in the next set of notes.

2.3 Finding Help

2.3.1 What's in a Library.

The answer is that many things can be contained in a library. The libraries are bundles of code used to conduct analysis, create graphs, etc. Libraries usually also contain some data which can be loaded and used for a library provided example. At this point, you're likely thinking, so what; how do I use what's in a library? Well, the bad news is that you need to know what is in a particular library before you can use it. The good news, of course, is that it's easy to discover what's in a particular library. This leads us to one of the many ways we can seek help in R. Working from previous notes, let's take a look at the base library which comes installed with the base installation of R and loads upon start up of the program R. In the console, type the following and hit the enter key:

```
help(base)
```

There are some things to notice in the new help window. First, at the top of the window; you'll see the topic you asked for help on (typically a function), the package in which this help query topic is found, and R Documentation which is where this help comes from. The key information for this particular library help is what the library does or what can it do; which is displayed in the details section; which begins "This package contains the basic functions which let R function as a language: arithmetic, input/output, basic programming support,...". Also note that this is a rare instance when the package help does not contain a list of the functions available in the package. However, it does tell us what to do to get the complete list of functions:

```
library(help="base")
```

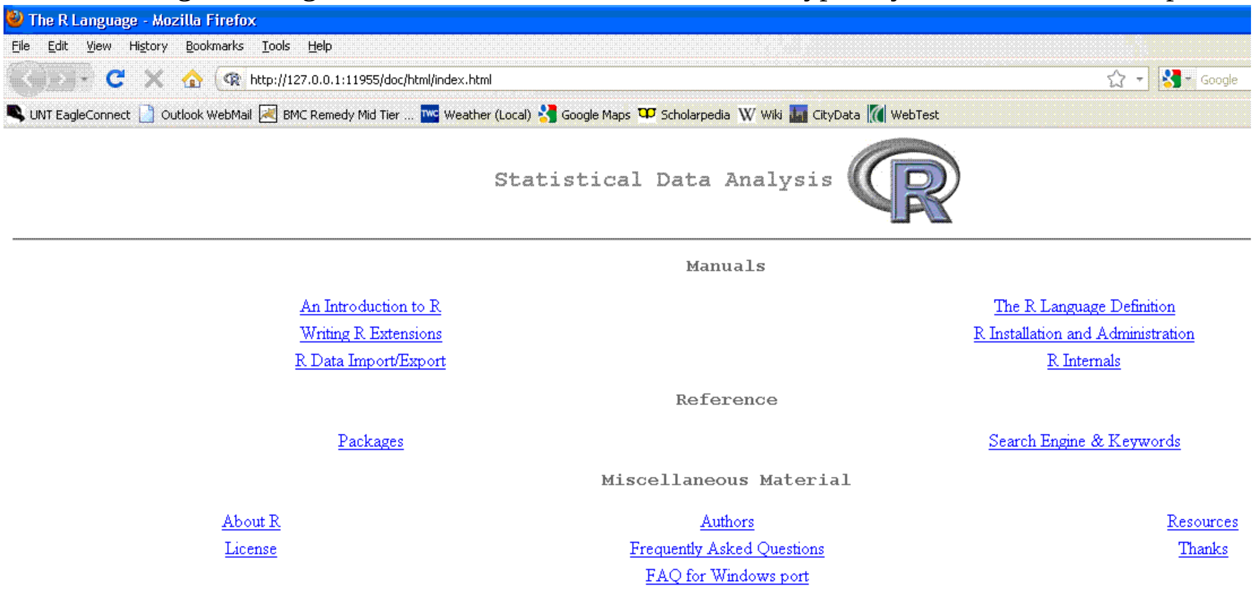
which does give us a complete list of functions and a brief description of their use. So, what is a function? Well, let's take a look at:

```
help(mean)
```

This help window is more typical of what you'll see when using the console help function. There are key elements here which appear in most help documentation; those listed in red (e.g. description, usage, arguments, details, etc.). Especially important is what we find at the bottom of the help window; the examples. All examples listed in these types of help windows can be copied and pasted into the console and they will work–and importantly can then be modified for our particular use. We will use this approach in later notes to see how to do a particular analysis and apply it to our own data.

2.3.2 Finding the right library.

If we are interested in finding a library that will allow us to do some task or analysis; then we have a multitude of choices for tracking down what library we need or want given the likelihood of multiple libraries able to do a given task. If we start by clicking on the 'Help' button in the task bar at the top of the console, we find a variety of help options. Two of my most frequently used strategies for finding help are (1) 'HTML help' which opens your default browser to the online R help index and (2) 'Search help...' which searches help files for whatever topic you enter. First, a note of caution; if you have all the available packages downloaded and installed, the 'Search help...' will take a minute or more to collect all the results for just about any topic you search, often resulting in a large list of returns. For this reason, I typically use the HTML help first,


The R Language - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://127.0.0.1:11955/doc/html/index.html
Google
UNT EagleConnect
Outlook WebMail
BMC Remedy Mid Tier ...
Weather (Local)
Google Maps
Scholarpedia
Wiki
CityData
WebTest

Statistical Data Analysis

Manuals

An Introduction to R
Writing R Extensions
R Data Import/Export
The R Language Definition
R Installation and Administration
R Internals

Reference

Packages
Search Engine & Keywords

Miscellaneous Material

About R
License
Authors
Frequently Asked Questions
FAQ for Windows port
Resources
Thanks

because I can click on the 'Search Engine & Keywords' to look for packages, functions, or other forms of information on a particular topic; or I can click on 'Packages' to review the packages' descriptions and then click on and review a particular package's documentation and related functions. There are other ways of finding help, often more efficient; such as searching Google[6] using "R xxxxxxx" where xxxxxxx is the topic of interest, or using the Rseek[7] search engine. Using Google will inevitably lead you to one of the many very useful blogs created by R users who, not long ago were in the exact same situation you might be...looking for help with some function or library. Also keep in mind there are several R Reference Cards available; I have this[8] one posted on the web page[9] because, I prefer it to others. Of course, there are also the help options in the R console: 'FAQ on R', 'FAQ on R for Windows', and the help 'Manuals (in PDF)' all of which should be considered recommended reading.

It is often intimidating to see how much help is available and realize finding what you want can become an adventure in and of itself. But, imagine how ridiculous someone might find it if we complained about having too much help available for a software package we were learning. ?.?.?

[6]http://www.google.com/
[7]http://www.rseek.org/
[8]http://cran.r-project.org/doc/contrib/Short-refcard.pdf
[9]http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/R_SC/

2.4 Keeping R Up-To-Date

Maintaining R is not difficult, it simply requires a few easy steps.

If you have your ‘Rprofile.site’ file setup to auto-load packages when R is started, then you will need to remove that file from the R directory prior to updating. If you do not, then the packages which you have specified to auto-load will never be updated. This is because R will not update a package if that package is loaded into the workspace. So, if you need to remove the ‘Rprofile.site’ file, then find the ’etc’ folder in your R installation. Two examples of where it might be located are below where x.xx.x refers to the version number of the R installation you have (e.g. R-2.15.0).

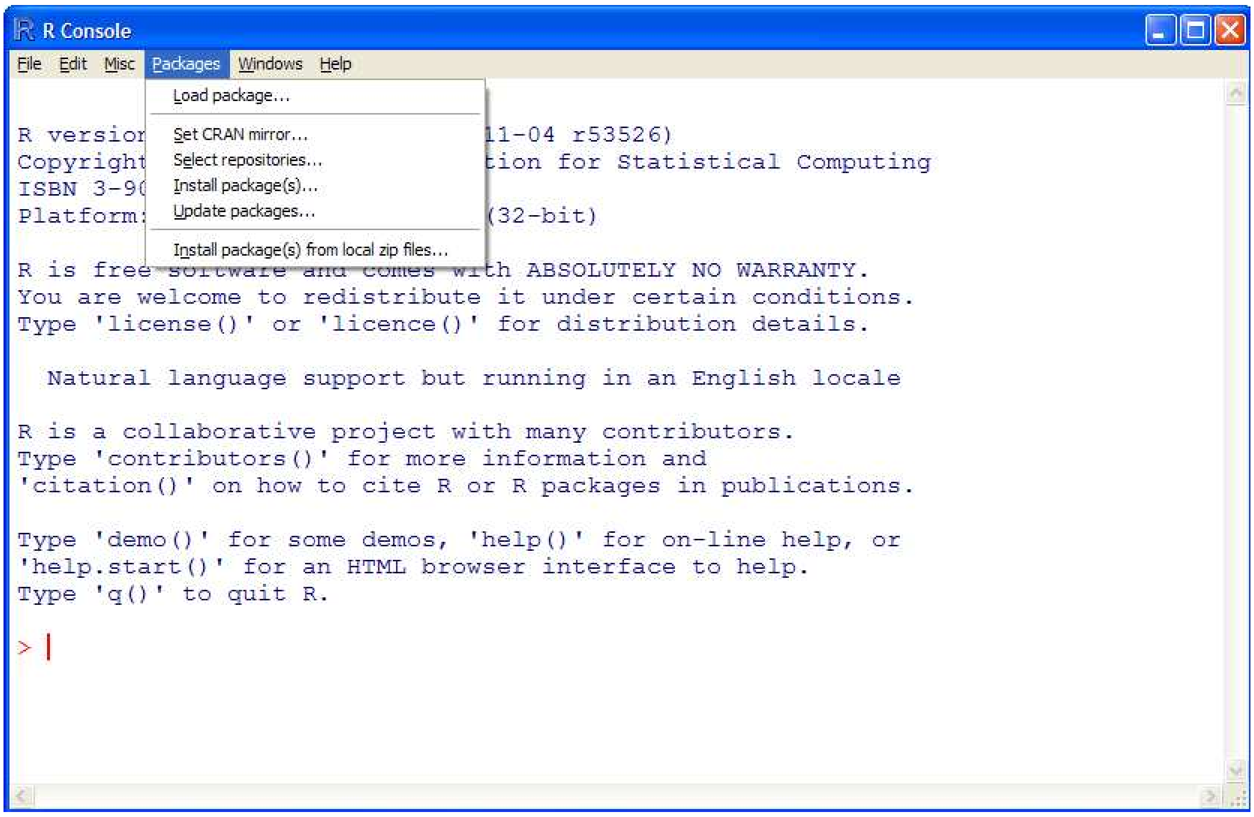
```
C:\R\R-3.0.1\etc
C:\Program Files (x86)\R\R-3.3.1\etc
```

Once inside the ‘etc’ folder, right-click on the ‘Rprofile.site’ file and select “Cut”. Now right-click on your desktop and select “Paste”. You do not need to close the ‘etc’ window/folder.

2.4.1 Installing new packages.

First, start R.

Second, click on “Packages” at the top of the R console and then select “Set CRAN mirror...”. Choose the mirror site you wish to use; I generally select USA (CA 1).



```
R versio                              11-04 r53526)
Copyright                             tion for Statistical Computing
ISBN 3-9(
Platform:                             (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

Next, click again on “Packages” in the R console and then select “Select repositories...”. By default, CRAN and CRAN (extras) are selected; I generally also select R-Forge which contains packages begin developed (often very recent updates to existing packages). Then click the OK button.

Next, type the following line in the R console and hit the enter key:

```
x <- new.packages()
```

This line tells R you want to search for new packages at the repositories selected above and if any are available, they will be assigned to the object "x". The reason for doing this is so you can then (once it finishes; it can take a few minutes), you can type "x" and hit the return key to see the names of the new packages; thus allowing you to decide if you want all of them, some of them, or none of them.

Next, if you would like to install all of the new packages; then simply type the following line in the R console and hit the enter key:

```
install.packages(x)
```

If you would like to install only some of the packages, then you need to reference which ones you want using the number associated with the packages you want. For example, perhaps you only want the 1st package listed (in "x") and the 5th, 8th, 9th, and 10th – then you would use the following script:

```
install.packages(x[c(1,5,8:10)])
```

Notice above, we used the brackets to refer to specific elements of "x" and we use the *concatenate* (or combine – "c") to further specify multiple elements of "x". You could simply use quotation marks around the names of each package listed in "x" instead of the numbers (1, 5, and 8 through 10).

2.4.2 Updating installed packages.

Next, click again on "Packages" in the R console and then select "Update Packages...". Like with the new packages command, this can take a few minutes. Once done searching for updates, you will be presented with a list of updated package, you can select all of them, some of them, or none of them as you see fit.

Congratulations, you now have the most up to date packages.

Now close R and return to the desktop, cut the 'Rprofile.site' file from the desktop and paste it back into the 'etc' folder. Close the 'etc' window/folder and you're on your way.

To determine if a new version of R is available, simply point your favorite browser to:

```
http://cran.r-project.org/bin/windows/base/
```

3 Module 3: Getting Data Into R

3.1 Script Files.

Recall that early on in these tutorial notes, it was mentioned that there are generally three windows you will use frequently in R: the console window, the graphics window, and the script window. The script window is not necessary; but often preferred for building script or code and proof reading it prior to submitting it (much like the syntax windows/editors found in SPSS and SAS). To open a new script window, simply click on 'File' then 'New script'. We can write as much script here as desired and highlight, right click, then submit individual elements or the entire script as necessary. Another benefit to using a script window comes when saving our work. Script files are extremely small (i.e. virtually identical in size to equivalently lined text files [.txt]) and, if appropriately thorough can be loaded and run to produce the entirety of our work from a given session. The alternative; is to save the script file and the workspace image (which is everything contained in the console); but, a workspace image can grow quite large and thus consume an often undesirable amount of space. The reason for my discussing script files here is that we will be using R Commander (Rcmdr) to import data but Rcmdr is not necessary to import data; only the script is necessary. In future tutorial notes, a script file (e.g. filename.R) will be all that is provided.

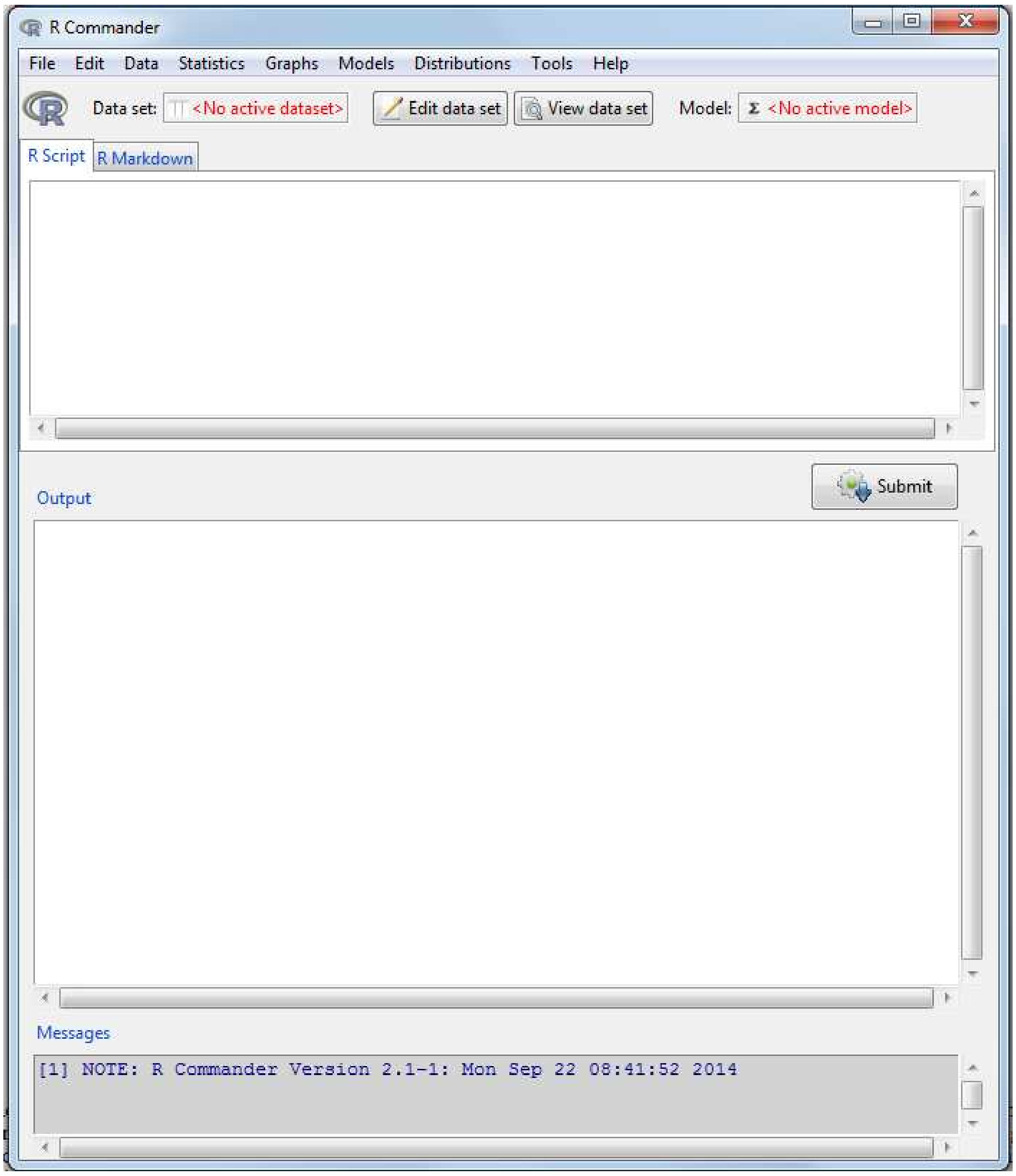
3.2 Initial Rcmdr orientation.

**Note: if you've been following these tutorial notes from the first, you should have downloaded, installed, and updated all the packages used on this site. Although this may seem excessive, some tasks in Rcmdr require other packages (and Rcmdr will load them as needed only if they have been downloaded and installed), so please take the time to get all the packages used on this site*[10].

For those of us (me included) who started out with point-and-click statistical software, Rcmdr represents a somewhat familiar interface for some basic tasks. It also provides script for each task specified through point-and-click operations; which allows us to see how we might graduate away from R Commander as we progress to more complex tasks not available in Rcmdr. So, let's get started by loading the Rcmdr package:

```
library(Rcmdr)
```

[10]`http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/R_SC/Module2/InstallPackages_lis`



```
[1] NOTE: R Commander Version 2.1-1: Mon Sep 22 08:41:52 2014
```

Right away; you can likely see that Rcmdr was made to be user friendly. Let's orient ourselves by starting at the bottom and working upward. At the bottom of Rcmdr we find a 'Messages' window that generally serves to let us know when something didn't go quite right; in other words error messages will appear here along with warning messages. Error messages will appear in red and reflect an error which prevented a function/task from being carried out. Warning messages (and note messages) will be displayed in blue and do not necessarily reflect a failure of a function/task to be carried out. The output window is as the name implies where all output will be displayed; with the exception of graphics which will be displayed in a separate window outside of Rcmdr. The script window, again as the name implies; displays script which result from specifying some task, analysis, or function through the use of the point-and-click menus. You can also write/type script directly into the Rcmdr script window and then highlight and submit it using the 'Submit' button between the script and output windows. As an example; type the following into the Rcmdr script window and then highlight and submit it:

```
x <- function
```

You will notice the script appears in red in the output window and no actual output (would be displayed in blue) appears. The reason no output appeared is because there was an error; displayed in the messages window at the bottom. The rest of the Rcmdr buttons and menu items will be fairly self explanatory; but, we will be using some of them here.

3.3 Using Rcmdr to import SPSS data.

For now; let's get some data imported. First, you need to download the example data files from the web page (Example Data 1[11], Example Data 2[12], & Example Data 3[13]) and save them to your source directory.

In Rcmdr, click on 'Data' and take note of the available choices. First, let's create a simple data file; so, click on 'New data set...' and then enter the name 'ex1' then hit the 'OK' button. This brings up the Data Editor where you can type in data values. Notice too, you can click on 'var1' and give the first variable a name, as well as specify it as numeric or string. For now; go ahead and close the data editor and close the 'New Data Set' naming window. Again; you will see some script, output, and an error stating "empty data set". If we again, click on 'Data' and then 'Load data set...' we could open an existing R data file. However, we generally want to open an existing data file that is not in an R format. But, before we do that; left-mouse click then right-mouse click in the Script Window and select 'Clear Window'. You can do this in the Output Window as well. To clear the Messages window, you need to highlight all the text in that window and then delete it.

In Rcmdr, click on 'Data' and then hold the cursor over the 'Import data'. We will import an SPSS file first, so click on 'from SPSS data set...'. Next, we will be prompted to name our data fileuse the name 'example1'. We also see that by default the value labels will be converted to factor levels and the maximum number of value labels for factor conversion is set to infinite. Once you have typed in the name (example1), click 'OK'. If you set your source directory correctly and you downloaded the example data sets into that source directory, you should be looking at them now. Highlight 'ExampleData1.sav' and then click the 'Open' button. Now, looking at the Script Window (and Output Window) you should see the appropriate script for importing an SPSS data file into R:

```
example1 <- read.spss("C:/Users/jons/Desktop
  /Work_Stuff/Jon_R/Example Data/ExampleData1.sav",
  use.value.labels=TRUE, max.value.labels=Inf, to.data.frame=TRUE)
```

All you would need to change for future use is the file name (and path if the data is not located in your source directory).

*You will need the 'foreign' library loaded if you are working with just the R console or from the R console and a script file (i.e. not using Rcmdr). This is why I have the foreign library listed in my Rprofile.site file as one of the libraries to load upon start up of R.

Notice also some key features of the script: we have created an object 'example1' and assigned it '<-' using the 'read.spss' function and our object was created as a data frame (an R

[11] `http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/R_SC/Module3/ExampleData1.sav`
[12] `http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/R_SC/Module3/ExampleData2.xlsx`
[13] `http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/R_SC/Module3/ExampleData3.txt`

way of saying or identifying a data file matrix). Also notice our specified options from above (e.g. use.value.lables=TRUE, max.value.labels=Inf); which are important as examples of the way R specifies conditions using =TRUE or =FALSE. Many functions use arguments like these conditional true/false statements as options for further specifying some task within the function. If you would like more information on the 'read.spss' function or you would like an example of using the console help; then type the following in the R console and hit enter:

```
help(read.spss)
```

You should also take note that we now have a current data set specified just above the script window in Rcmdr. Therefore, we can click on the 'Edit data set' button to edit the data or we can click on the 'View data set' button to view it—both of which produce script: `fix(example1)` and `showData(example1, ...)`. Now, close both the data view window and the data editor window if you have not done so already. An extremely common practice when working with script is to use # to comment out anything that is not used as working script (i.e. notes, reminders, comments, etc.). For example, type (or copy and paste) the following in the Script Window of Rcmdr:

```
# This is how we view a data set loaded in Rcmdr.
showData(example1, placement='-20+200', font=getRcmdr('logFont'),
  maxwidth=80, maxheight=30)
```

Now, highlight those two lines and click the 'Submit' button between the Script Window and the Output Window in Rcmdr. Next, close the data view window and then copy and past those two lines into your R console. Next, close the data view window again and then, in the R console, click on 'File' and 'New script'. Now, copy and paste those two lines into the new script window you have just opened. Next, highlight those two lines in the new script window and then right-mouse-click on the highlighted text and select 'Run line or selection'. Now go ahead and close the data view window for a final time.

3.4 Using Rcmdr to import Excel data.

Click on 'Data' and then hold the cursor over the 'Import data'. Then click on 'from Excel, Access, or dBase data set...'. Next, we will be prompted to name our data file—use the name 'example2' and click 'OK'. Highlight the 'ExampleData2.xlsx' and click 'Open'. It's just that simple. One thing to note is that now when you click on the 'Data set: example2' button in Rcmdr, above the Script Window; you can now select which data set you would like to use (example1 or example2). Meaning, you can have multiple data sets available during a single session and simply switch between them as necessary.

3.5 Using Rcmdr to import text data.

Click on 'Data' and then hold the cursor over the 'Import data'. Then click on 'from text file, clipboard, or URL...'. Next, we will be prompted to name our data file—use the name 'example3' and notice all the options for specifying the nature of the data. None of these default options needs to be changed with this file, so click 'OK'. Highlight the 'ExampleData3.txt' and click 'Open'. Again, it's just that simple.

You will now notice that with a data file loaded into Rcmdr you can click on the different menu options and a variety of functions, analysis, graphs, etc. are only a mouse–click away. Also remember, virtually any script generated in Rcmdr can be used in the R console with the necessary libraries loaded.

3.6 Importing data Without Rcmdr

Rcmdr is not necessary (as mentioned above), one can simply import data directly using the proper script (e.g., read.table & read.spss). If we were starting a new session of R, close R and then re-open the program, we could open a script window by clicking on 'File', then 'New script' in the R console. In this new script window, type the following to import the ExampleData3.txt file:

```
example.3 <- read.table("http://bayes.acs.unt.edu:8083/
     BayesContent/class/Jon/R_SC/Module3/
     ExampleData3.txt", header=TRUE, sep="",
     na.strings="NA", dec=".", strip.white=TRUE)
summary(example.3)
```

In the script window, type the following to import the ExampleData1.sav file:

```
library(foreign)
example.3 <- read.spss("http://bayes.acs.unt.edu:8083/
     BayesContent/class/Jon/R_SC/Module3/
     ExampleData1.sav", use.value.labels=TRUE,
     max.value.labels=Inf, to.data.frame=TRUE)
summary(example.3)
```

That's it, you simply need to have the 'foreign' library loaded in order to use the 'read.spss' function. If you have data on your machine and want to import it using a browse function, simply replace the `"http://www.webaddress.com"` with `file.choose()` in each of the above read functions (i.e., read.table & read.spss).

3.7 Importing many Excel files, each with multiple sheets.

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, DSA personnel do not recommend using Excel; for data storage, data display, or data analysis. An often quoted phrase[14] 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). DSA recommends storing data in plain text (.txt) files with comma delimiters; also known as a comma separated values (.csv) file type. The reason DSA 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**[15].

[14] 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.

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

3.7.1 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 DSA servers so that the reader can download the data files[16] and replicate what is illustrated below. Our goal was to import all the data and merge it into a single data frame.

3.7.2 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)
```

[16]The data can be downloaded from the following links:
http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/Benchmarks/ExcelFiles/participant.1
http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/Benchmarks/ExcelFiles/participant.2
http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/Benchmarks/ExcelFiles/participant.3
http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/Benchmarks/ExcelFiles/participant.4
http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/Benchmarks/ExcelFiles/participant.5
http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/Benchmarks/ExcelFiles/participant.6
http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/Benchmarks/ExcelFiles/participant.7
http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/Benchmarks/ExcelFiles/participant.8
http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/Benchmarks/ExcelFiles/participant.9
http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/Benchmarks/ExcelFiles/participant.1

```
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 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 = TRU
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 jth 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 jth 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 kindergartenalways 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.   :-501.6
participant.10.xls: 3000   Dim   :10000   1st Qu.: 250.8   1st Qu.:-249.7
participant.2.xls : 3000   Off   :10000   Median : 500.5   Median :   0.0
participant.3.xls : 3000                  Mean   : 500.5   Mean   :  -1.0
participant.4.xls : 3000                  3rd Qu.: 750.2   3rd Qu.: 247.1
participant.5.xls : 3000                  Max.   :1000.0   Max.   : 501.5
(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)
```

3.7.3 Conclusions on importing many Excel files

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[17] 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[18]. 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[19] which was used.

4 What's next.

This concludes the first 3 modules of the DSA Do it Yourself (DIY) Introduction to R short course. In future tutorial notes, we will be using R console and script files exclusively; but remember all scripts can be copied and pasted into the Script Window of Rcmdr; or simply loaded into the Script Window using 'File', 'Open script file...' in the Rcmdr top task bar.

When reading the script files, you'll notice the common convention of using # to start a comment line (which is not working code), while lines without # are working code. So, without further ado, onward; TO INFINITY AND BEYOND..., or perhaps just Module 4 and some Initial Data Processing:

`http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/R_SC/Module4/M4_InitialProc`

References & Resources

[17] `http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/R_SC/Module3/M3_ImportManyExcel.`
[18] `http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/R_SC/`
[19] `http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/R_SC/Module3/MultiExcelDataCreat`

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

This document was created using LaTeX