John Miyamoto (jmiyamot@uw.edu)
http://faculty.washington.edu/jmiyamot

January 5, 2014

Getting Started with R under Windows 7

NOTE: R is a free open-source statistical program. See the R homepage (http://www.r-project.org/) for the terms of its use. The homepage also has useful information about the history of the R-project, how to use R, how to write R programs, and much more. The purpose of this document is to describe a way to configure R that makes it easy to work with R on many different projects (at least, so it seems to me). .

NOTE TO MAC USERS: These instructions are written for Windows users because this is the only operating system with which I am familiar, but most of the information can be adapted (I believe) to a Mac or Linux framework. The following documents has information about installing and running R on a Mac:

- R for Mac OS X FAQ (http://cran.r-project.org/bin/macosx/RMacOSX-FAQ.html)
- R:Installing and Using R and Rcmdr on a MAC (http://wiki.math.yorku.ca/index.php/R:Installing_R_and_Rcmdr_on_a_MAC)

Table of Contents

1. Downloading R

These instructions assume that R version 2.15.3 is the current state-of-the-art version of R for Windows. If there is a more recent version, you should use it. To download R:

1.1. Go to **`http://www.r-project.org/`**.

1.2. Click on CRAN (Comprehensive R Archive Network) on the left (under "Download, Packages"). CRAN will ask which of various mirrored website you want to be connected to. Choose one that is close to where you are, e.g., the Fred Hutchinson website is in Seattle.

1.3. Under "Download and Install R" (precompiled binary versions), click on the operating system for your computer, e.g., MacOS or Windows. DO NOT download the R source code unless you are a computer expert who wants to work with the code. The average user does not look at this code.

1.4. For either MacOS or Windows, there are two components to the program. **`Base`** refers to the main program. **`Contrib`** refers to special sets of functions (R calls them "packages") that have been written by R-users for special purposes. Everyone needs to have the **`Base`** program, but you only need a package if you need the special purpose functions that are contained in the package. E.g., there is a package called **`bootstrap`** that has functions for computing bootstrap statistics; you should install this package only if you need to compute bootstrap estimates. Section 3 describes how find out what packages are available and how to install them.

1.5. Download the latest version of R for your operating system (Windows; Mac; or Linux). This is all you need to install the R base program. Section 2 describes how to install R.

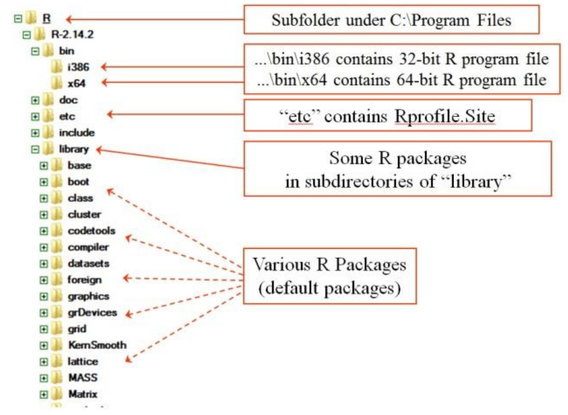
In addition, you may want to download some documentation. Starting from CRAN (http://cran.us.r-project.org/), notice that the left side of the screen has links to documentation.

1.6. To download the R Manuals (see Section 10), click on the Manuals link. This will take you to a page from which you can download manuals for different purposes.
1.7. Many R users have contributed useful documents that describe different applications of R. Click on the Contributed link (from CRAN). Look for documents that fit your needs.
1.8. It may be useful to look at the FAQ for Windows and Mac users (again, see Section 10). Go to http://cran.r-project.org/, and click on FAQs. Choose the R Windows FAQ or the R MacOS X FAQ.

I'm afraid that I don't use a Mac, Unix or Linux systems, so I don't have tips about what to download for these operating systems, but I believe that the procedure is reasonably straightforward (I haven't heard people complaining about it).

2. Installing R on a Windows System

The files README and RW-FAQ documents contain instructions for installing R. To install R, simply apply the Add/Remove programs utility (available on the Windows Control Panel) to the R installation file (named something like **`R-2.15.3-win32.exe`**). Figure 1 shows the directory structure that is automatically created when R is installed. The R directory is itself a subdirectory of **`C:\Program Files\`**. On your computer, your version of R may be installed in a directory that is called something like, **`C:\Program Files\R\R-2.15.3`**. In any case, the R program will have a directory structure that is fairly similar to the structure shown in Figure 1.


R
R-2.14.2
bin
i386
x64
doc
etc
include
library
base
boot
class
cluster
codetools
compiler
datasets
foreign
graphics
grDevices
grid
KernSmooth
lattice
MASS
Matrix
Subfolder under C:\Program Files
...\bin\i386 contains 32-bit R program file
...\bin\x64 contains 64-bit R program file
"etc" contains Rprofile.Site
Some R packages in subdirectories of "library"
Various R Packages (default packages)
Figure 1

A Brief Comment on 32-Bit R versus 64-Bit R

The current Windows computers have either a 32-bit or 64-bit processor (never both). A 64-bit processor can usually (not invariably) run software that was written for a 32-bit processor, but the opposite is not true. To find out whether your computer has a 32-bit processor or a 64-bit processor, go to:

http://windows.microsoft.com/en-us/windows7/32-bit-and-64-bit-Windows-frequently-asked-questions

The standard installation of R installs both a 32-bit version of R and a 64-bit version of R. If you are using a computer with a 32-bit processor, you can only use the 32-bit version of R, but if you are using a computer with a 64-bit processor, you can use either version of R. The following link will take you to a discussion of the choice between the 32-bit and 64 bit versions of R:

http://cran.r-project.org/bin/windows/rw-FAQ.html#Should-I-run-32_002dbit-or-64_002dbit-R_003f

From the standpoint of the user, the main differences between the 32-bit and 64-bit versions of R are:

- If you have a 64-bit processor, then the 64-bit R will generally be better (slightly faster) than the 32-bit R.
- There are a few R packages that only have a fully function 32-bit version. For example, the BRugs package which is used to interface R with OpenBUGS runs reliably on the 32-bit version of R. There is a beta version of BRugs for the 64-bit version of R, but it may not always work (I have experienced frozen screens with BRugs on the 64-bit version of R that went away when I switched to using BRugs with the 32-bit version of R)[1].
- There is a separate **`Rgui.exe`** program (runs the R interface) for the 32-bit and 64-bit versions of R. For 32-bit R, **`Rgui.exe`** is in the i326 folder (see Figure 1) and for 64-bit R, **`Rgui.exe`** is in the x64 folder (also, see Figure 1).

R-code runs equally well and without any changes on the 32-bit and 64-bit versions of R, except when R is used in combination with one of the few packages that are not yet fully compatible with 64-bit R. My basic advice is to use the 64-bit version of R if you have a 64-bit processor, but be aware that some packages, in particular, BRugs, may require that you temporarily switch to the 32-bit R.

Mac and Linux users can disregard these comments because they are mainly relevant if you want to use the BRugs package as an interface between R and OpenBUGS; OpenBUGS will not run under Mac[2] or Linux OS.

Creating a Desktop Icon for R

After the program has been installed, you will want to put an icon (short-cut) to the **`Rgui.exe`** program file on your desktop. **`Rgui.exe`** for 32-bit R is contained in the **`...\R Version#\bin\i386`** directory, and the **`Rgui.exe`** for 64-bit R is contained in the **`...\R Version#\bin\x64`** directory. **`Rgui.exe`** runs the interactive version of R. Right mouse click on the **`Rgui.exe`** to create a shortcut to R, and then copy or cut and paste this shortcut to the Windows Desktop (or any other place you might want to keep the icon).

3. Installing R Packages on Any System (Windows, MacOS, Linux)

WARNING: Under Windows 7, you may need to establish that R has "administrator privileges" before R is allowed to add new packages to the R installation. I don't know the principles behind how this works, but here is one way to give administrator privileges to your installation of R: Right-click on the R program icon (on the Desktop) and choose "Properties". Then go through every tab of the dialog box, click on every button that is labeled "Advanced", and if there is any option to "run as administrator", choose that option. I find that this generally works, but I don't really understand what is happening here.

[1] According to the R documentation, the main example of packages with only 32-bit versions are **`BRugs`** and **`rggobi`**. Actually **`BRugs`** has a beta 64-bit version.

[2] Mac users can run R and OpenBUGS in a simulated Windows environment, i.e., by using Parallels, VMWare Fusion, BootCamp, etc. If you plan to do this, then you may want to install BRugs on the Windows partition.

You may also want to install some R packages. As a matter of fact, the default installation automatically installs a number of packages, e.g., "base", "boot", "class", "cluster", ... To see a list of packages, return to the CRAN page (**`http://cran.us.r-project.org/`**); click on Packages in the Software section of the navigation panel. You should see a list of available packages. As a practical matter, most users learn about useful R packages because they are mentioned in books, journal articles or conversations with friends. Looking through the package list at CRAN can be useful, but it is time consuming and not always informative.

If your computer is currently connected to the internet, you can download and install an R package as follows. Run R, and type the R command:

```
install.packages(c(<package names>))
```

where **`<package names>`** is replaced by a quoted series of package names. For example, if you want to install the **`bootstrap`** and **`survival`** packages, you would enter the command:

```
install.packages(pkgs = c("bootstrap", "survival"), repos = "http://cran.fhcrc.org/")
```

The expression, **`c("...", "...", ...., "...")`** specifies a series of packages (character vector of package names) that will be installed on your system. The **`repos`** argument specifies a computer repository where the R packages can be found. The specification **`repos = "http://cran.fhcrc.org/"`** specifies the Fred Hutchinson computer in Seattle, which is a convenient repository for people living in Seattle. To find a repository that is close to your current location, run R; then click on the "Packages/Select Repositories" menu. The **`install.packages`** method is the easiest way to install R packages[3,4].

Even after you have installed a given package, e.g., the **`bootstrap`** package, you still must give the command, **`library("package.name")`**, where "package.name" is the name for the particular package, to make the package available to your current R session. For example, to load the **`bootstrap`** package, you must give the R command, **`library(bootstrap)`**. Of course, this only works if you previously downloaded and installed the **`bootstrap`** package. Below is a list of some of the many useful R packages.

- The **`foreign`** package (foreign.zip) contains a function, **`read.spss`**, that allows R to read SPSS data files.
- The **`MASS`** package (vr.zip) contains many useful functions and data sets for Venables & Ripley's excellent textbooks about S computing (also R). This package is automatically installed when you install the base package (the main R program), so you don't need to download it or install it individually.
- The **`NLME`** package (nlme.zip) contains functions and data for repeated measures anova and multilevel modeling. This package is automatically installed when you install the base package (the main R program), so you don't need to download it or install it individually.
- The **`lattice`** package (lattice_0.14-16.zip) contains functions for high quality graphics.
- The **`BRugs`** and **`R2OpenBUGS`** packages contain functions for getting R to work in concert with the OpenBUGS program.

[3] The 'install.packages' method that is described in this document assumes that you want to install the packages in the default locations for the R program. It is possible to install the packages in other user-specified locations - see 'help("install.packages")' for documentation about how to do this.

[4] It is also possible to install packages by running R; then click on the "Packages" menu; the click on "Install packages"; then click on the name of a package that is displayed on the resulting menu.

- The **`rjags`** and **`R2jags`** packages contain function for getting R to work in concert with the JAGS program.
- The **`R2WinBUGS`** package contains functions for getting R to work in concert with the WinBUGS program.
- Note: It is very easy to update R packages, i.e., get the latest versions of packages that are already installed on your computer. To update R packages, do the following: (i) connect to the internet; (ii) run R; (iii) give the command, **`update.packages()`**, to R. The **`update.packages`** function will automatically update every package that you have installed on your computer.

4. Setting the Startup Directory for R TOC

The startup directory is the directory that contains the data objects that you want to work on in your current R session. For example, suppose you are working on two projects; the data files for your dissertation project are in a directory (folder) called **`c:\diss\data`** and the data for your job are in a directory called **`c:\job\data`**. In general, it is easier to load or save data from files in the R startup directory than from files in other directories. Therefore, when you use R to analyze your dissertation data, you want to start R with **`c:\diss\data`** as the startup directory; and when you use R to analyze data from your job, you want to start R with **`c:\job\data`** as the startup directory.

- To set the startup directory for R, click the right mouse button on the R program icon. Choose "properties". Click on the Shortcut tab. In the "Start In" field, type the path and name of the directory that contains the files that you want to work on. Then click OK. This sets the startup directory for this particular icon. You can choose any directory as the startup directory, and you can create multiple icons for different projects.
- *Example:* Suppose as suggested above that you are working on two projects, a dissertation project with files in **`c:\diss\data`**, and a job project with files in **`c:\job\data`**. A standard way to deal with this is to make two R program icons. Right mouse click on any existing R program icon or shortcut; select Create Shortcut; do this twice if you want two shortcuts to R. Now right click on an icon, select the properties of the icon, and edit the properties of these icons, setting one so that it starts in **`c:\diss\data`** and the other so that it starts in **`c:\job\data`**. Click on the appropriate icon when working on the corresponding project. In this way, R objects that are created for purposes of analyzing the dissertation data (in **`c:\diss\data`**) will not get confused with R objects that are created for purposes of analyzing job data (in **`c:\job\data`**).

Suppose you want to work on the data in **`c:\diss\data`**. Double click on the R icon that starts in **`c:\diss\data`**.

- As it starts up, R automatically looks for a file called **`.Rdata`** in **`c:\diss\data`**. This file, **`.Rdata`**, contains any functions or data objects that were created and saved in previous R-sessions that started in **`c:\diss\data`**. If there is no **`.Rdata`** file in **`c:\diss\data`**, e.g., because this is the first time you have started R in this directory, then R automatically creates a **`.Rdata`** file.
- When you initially start R, the **`.RData`** file in the startup directory is loaded into the computer's working memory and is given the name, **`.GlobalEnv`**. **`.GlobalEnv`** can be thought of as the workspace for your current R session. If you create additional objects during your R session, these objects will be added to **`.GlobalEnv`**, but they will not be permanently saved until you issue a command that saves the workspace. You can also use the function, **`save.image`**, to save the current workspace to **`.Rdata`** (see the R-Help page for the **`save.image`** and **`save`** functions).

- In addition, when you quit R, you will be queried whether you want to save your current workspace to **`c:\diss\data`**. If you choose to save your current workspace, then all functions and objects in your current workspace will be saved to the **`.Rdata`** file in **`c:\diss\data`**. These functions and objects will be loaded the next time you start R in **`c:\diss\data`**.

5. How does R find functions and objects (e.g., variables or data sets) that are referenced in the code for an analysis?

Whenever you enter an R-expression at the R-prompt, R looks for functions or objects that correspond to the functions or objects that are referred to in the expression. For example, suppose you type the following at the R-prompt (the initial "**>**" is the R-prompt):

```
>X = Y + Z
```

R will look for objects called "**`Y`**" and "**`Z`**", and try to assign to "**`X`**" the sum of the values assigned to "**`Y`**" and "**`Z`**". To find **`Y`** and **`Z`**, R looks through a series of files for objects with these names. This series of files is called the *search path* for R. You can see the current search path by giving the R command, **`search()`**. For example on my computer, **`search()`** produces the information shown in Table 1.

Table 1

```
> search()
[1] ".GlobalEnv"         "file:data.rda"       "file:e:/r/jm.utilities.rda"  "package:stats"
[5] "package:graphics"   "package:grDevices"   "package:utils"               "package:datasets"
[9] "package:methods"    "Autoloads"           "package:base"
```

This output tells us that **`.GlobalEnv`** is in position 1 (the top position on the search path). The files in positions 2 - 11 are occupied by other files that contain useful functions. Position 11 contains the base package which constitutes the primary functions of the R programming language. The specific search path will differ on different computers, or even on the same computer at different times; later I will explain how the user can control which files will be placed on the search path.

If the expression, "**`X = Y + Z`**", is entered at the R-prompt, R looks for objects corresponding to "**`Y`**" and "**`Z`**". R looks first in **`.GlobalEnv`**, second in the file **`data.rda`**, third in the file **`jm.utilities.rda`**, fourth in the file **`package:stats`**, etc. R assigns to each expression the first object with a matching name that it finds on the search. For example, suppose that there is an object named "**`Y`**" in **`data.rda`** where "**`Y`**" has the value 4, and there is another object named "**`Y`**" in **`package:datasets`** where it has the value 10, and there is no object named "**`Y`**" in any other file along the search path. **`Y`** will be assigned the value 4 because this is the value of **`Y`** in the earliest file on the search path. Suppose that there is an object named "**`Z`**" in **`.GlobalEnv`** where it has the value 2, and there is another object named "**`Z`**" in **`package:stats`** where it has the value 15, and there is no object named "**`Z`**" in any other file along the search path. Then Z will be assigned the value 2 because this is the value of Z in the earliest file on the search path. The expression "**`X = Y + Z`**" results in assigning the number 6 to "**`X`**" (because 6 = 4 + 2). Furthermore the object called **`X`** will be placed in **`.GlobalEnv`** (if there previously existed an object called **`X`** in **`.GlobalEnv`**, that object would be replaced by the **`X`** whose value is 2).

In general, whenever an R-expression is entered at the R-prompt, R looks for functions and objects that correspond to functions and objects that are referenced in the expression. R will always interpret references according to the first function or object of the same name[5] that occurs along the search path.

The **`attach`**, **`library`**, and **`require`** functions can be used to place sets of R objects on the search path. The **`detach`** function is used to remove files from the search path. Suppose that **`c:\diss\data`** is the startup directory, and you want to access the objects in a file, **`c:\rfiles\previous.rda`**. You can place it in position 2 of the search path with the function, **`attach("c:/rfiles/previous.rda", pos=2)`**. Note that the "\" symbol must be changed to a "/" symbol when indicating directory paths to R[6]. Once **`c:\rfiles\previous.rda`** is attached to the search path, then functions and other objects in **`previous.rda`** can be used in the current R program. The **`library`** and **`require`** commands are used to attach packages to the search path. See the online documentation for further information about **`library`**, **`require`** and **`detach`**.

6. How to attach files to the search path; how to detach files from the search path; how to create files of R objects TOC

How to create a file of R objects:

Suppose you have created three R objects, a vector **`X`**, a matrix **`M`**, and a function **`Func`**. You want to save them to a file from which they can be accessed at a future time. Here is one way to do it.

Table 2

# R Code	# Explanation

`X = c( 2, 4, 6 )`
`X`
 	
`M = matrix( 1:12, ncol = 4)`
`M`
 	
`Func = function( k ) return( 2*k )`
`Func( 3 )`
	Create a vector **`X`**, a matrix **`M`**, and a function **`Func`**.
	Then display these objects.

`save(list = c("X", "M", "Func"),`
`    file = "C:/r/data/newstuff.rda")`
	The vector **`X`**, matrix **`M`**, and function **`Func`** are saved to the **`C:\data\newstuff.rda`**[7].

There is a problem with the method shown in Table 2. If you create a new object **`Y`** and save it to the same file, i.e., **`save(list = c("Y"), file = "C:/data/newstuff.rda")`**, the second **`save`** command will overwrite the first **`save`** command, i.e., **`newstuff.rda`** will contain only the object **`Y`**. What we really want to do is to add the object **`Y`** to the other objects, **`X`**, **`M`** and **`Func`** that are already in **`newstuff.rda`**. Here is how to do this.

---

[5] Recent versions of R allow functions and data objects to have the same name without ambiguity. Thus, when R encounters new.name(x = 12), R knows that new.name refers to a function named "new.name" because the parentheses could only follow a function. When R encounters, y = x + new.name, R knows that "new.name" refers to a data object and not a function because it does not make sense to add a number to a function. R permits a function and data object to have the same name because it is always clear from context whether the name designates a function or data object. Thus if a variable **`X`** is in **`.GlobalEnv`** (position 1) and a function named **`X`** is in **`data.rda`** (position 2), R will know that **`X(y)`** refers to the application of the function **`X`** to the variable **`y`** because the expression clearly shows that **`X`** must be a function.

[6] An alternative is to give the command, attach("c:\\rfiles\\previous.rda", pos=2). In general, Windows directory specifications of the form "c:\aa\bb" must be converted to an R specification of the form "c:/aa/bb" or "c:\\aa\\bb".

[7] Reminder: "**`C:/data/newstuff.rda`**" is R notation for the file that Windows would call "C:\r\data\newstuff.rda".

Attaching a file to the search path

Suppose the file **`C:/data/newstuff.rda`** has already been created. We can put it on the search path with the **`attach`** function in R.

Table 3

# R Code	# Explanation
`attach("C:/data/newstuff.rda", pos = 2)`	C:/data/newstuff.rda") is placed on the search path in position 2.
`search()`	See the output of the **`search()`** function below.

```
 [1] ".GlobalEnv"                  "file:C:/data/newstuff.rda"  "file:data.rda"
 [4] "file:e:/r/jm.utilities.rda" "package:stats"              "package:graphics"
 [7] "package:grDevices"          "package:utils"              "package:datasets"
[10] "package:methods"            "Autoloads"                  "package:base"
```

Table 4.

# R Code	# Explanation
`ls(pos = 2)`	Shows the objects in position 2 of the search path. See output below.

```
[1] "Func" "M"    "X"
```

Using the **`move`** function to move additional objects into **`newstuff.rda`**.

The file **`jm.utilities.rda`** is a set of functions that I have created for my own computing needs. If you have not already done so, download **`jm.utilities.rda`** from http://faculty.washington.edu/jmiyamot/downloads.htm (there are also other places on my website where I keep a copy of this file so you may have downloaded it from, e.g., my course website). Once you have **`jm.utilities.rda`** on your computer, attach it to the search path with:

```
attach("C:/data/jm.utilities.rda")
```

You may have to modify the directory name where you keep **`jm.utilities.rda`**.

One of the functions in **`jm.utilities.rda`**, the **`move`** function, lets me move objects from one file on the search path to another file on the search path. For example, the following code creates a vector **`Y`**, and moves it to **`newstuff.rda`**.

Table 5

# R Code	# Explanation
`Y = c( 10, 20, 30)`	Create **`Y`**.
`move(Y, to = "newstuff.rda")`	Copies **`Y`** from **`.GlobalEnv`** to **`newstuff.rda`** and then deletes it from **`.GlobalEnv`**[8].
`ls(pos = 2)`	The **`ls`** command shows that **`Y`** is now in **`newstuff.rda`**.

```
[1] "Func" "M"    "X"    "Y"
```

Now **`newstuff.rda`** contains **`Y`** as well as the objects that were previously in this file, namely, **`Func`**, **`M`** and **`X`**. The **`move`** function is explained more fully in a separate document.

[8] In addition to moving **`Y`** to **`newstuff.rda`** on the R search path, the **`move`** command has to save the copy of **`newstuff.rda`** on the search path back to the file on the computer hard drive.

You can also delete (remove) objects from a file on the search path with the **`rm.sv`** function (also in **`jm.utilities.rda`**). E.g.,

Table 6

# R Code	# Explanation
`rm.sv("X", pos = 2)`	Removes **`X`** from **`newstuff.rda`**[9].
`ls(pos = 2)`	

```
[1] "Func" "M"    "Y"
```

Detaching a file from the search path

Table 7

# R Code	# Explanation
`search()`	Displays the current search path (see below).

```
 [1] ".GlobalEnv"                "file:C:/data/newstuff.rda" "package:graphics"
 [4] "package:grDevices"         "package:utils"             "package:datasets"
 [7] "package:foreign"           "package:stats"             "file:data.rda"
[10] "file:e:/r/jm.utilities.rda"      "package:methods"           "Autoloads"
[13] "package:base"
```

Table 8.

# R Code	# Explanation
`detach(pos = 2)`	Detaches **`newstuff.rda`**.
`search()`	Displays the current search path (see below).

```
 [1] ".GlobalEnv"           "package:graphics"     "package:grDevices"
"package:utils"
 [5] "package:datasets"     "package:foreign"      "package:stats"
"file:data.rda"
 [9] "file:e:/r/jm.utilities.rda" "package:methods"      "Autoloads"
"package:base"
```

There are many other uses for the **`move`** and **`rm.sv`** functions and other functions in **`jm.utilities.rda`**, but these will be discussed elsewhere.

The information below this line is not needed when you start to learn R. It is useful if you use R often.

----

7. Setting the Startup Configuration of R

The startup configuration is the way that R is set to work when you first start the R program. R has a default startup configuration, but experienced R users often prefer settings that are different from the default startup configuration. If you use R often, you will probably want to change the startup

[9] In addition to removing **`Y`** from **`newstuff.rda`** on the R search path, the **`rm.sv`** command has to save the copy of **`newstuff.rda`** on the search path back to the file on the computer hard drive.

configuration to your preferred settings so that you won't have to change R to your preferred settings every time that you start R. This section describes how to set these defaults. See Section 10.8 ("Customizing the environment") of *An Introduction to R*[10] for a fuller, and no doubt more accurate description of the initialization process for R.

When R starts up, it looks for instructions as to how to configure itself. It expects to find these instructions in a particular sequence of locations that are summarized in the following table.

Table 9. Files and functions that are run automatically during R startup.

Order	File or Function	What and Where
1	`Rprofile.site`	`Rprofile.site` is a text file in the `...\etc\` directory of your hard drive. The symbol "`...`" refers to the directory that contains the R program; you can find this directory either by inspecting the hard drive of your computer with a file utility, or by giving the R command[11]: `normalizePath(Sys.getenv("R_HOME"), win = "/")` The `Rprofile.site` file contains R commands that are automatically executed every time that a user starts R on this computer.
2	`.Rprofile`	`.Rprofile` is a text file in the user's home directory. The user's home directory can be found by giving the R command[12]: `Sys.getenv()["HOME"]` There may not exist a file called `.Rprofile` in the user's home directory, but if it does not exist you can create it (it must be a text file). Any R commands in `.Rprofile` will be executed during the R startup process.
3	`.First`	After executing the commands in `Rprofile.site` and `Rprofile`, R looks for a function called `.First` that may be stored in one of the positions along the search path[13]. If it finds such a function, it executes the `.First` function. If there is no `.First` function along the search path, then no `.First` function will be executed. If there are several `.First` functions at different positions on the search path, then the `.First` that is in the highest priority location, e.g., in `.GlobalEnv`.

Thus, the user can create a personal preferred configuration of R either by putting commands that configure R in the `Rprofile.site` file, or in the `.Rprofile` file, or by putting these commands inside of a `.First` function that resides on the initial search path[14].

There are a number of different ways to set the initial configuration of R. I will recommend one method because it is simple and flexible. First, you must decide where you want to keep all of your general purpose R functions. Let us suppose that you have decided to keep your R functions in a directory called `C:\r\data`. Step 1 involves creating a `.First` function, and putting it in a file in `C:\r\data`. (It is easy to change the location of your R functions, so make a temporary decision now; you can change it later.)

[10] *An introduction to R* is a pdf manual that can be downloaded from http://cran.r-project.org/doc/manuals/R-intro.pdf.
[11] On my computer, `Sys.getenv("R_HOME")` produces the output "C:/PROGRA~1/R/R-30~1.0". This is a computerese abbreviation for the path "C:/Program Files/R/R-3.0.0". On a Windows 7 computer, either string specifies the path to the same directory.
[12] On my computer, `Sys.getenv()["HOME"]` produces the output `"C:\\Users\\John M. Miyamoto\\Documents"`.
[13] See Section 5 for an explanation of what is the "search path."
[14] How to do this will be explained below.

Step 1: Run R. The following code creates a **`.First`** function that is a simplification of the one I use (the function is shown on the left; an explanation of the function is shown on the right):

Table 10. Create a **`.First`** function.

# R Code	# Explanation
`.First = function() {`	Begins the definition of the **`.First`** function.

`#   set the default help type`
`#   options(help_type="text")`
`    options(help_type="html")`
	Do you prefer help documentation formatted as text or as a web document (html) in a browser? I have set the option of html.

`    options(continue = "& ", digits = 8)`	Sets the continuation character to & and the default number of digits after the decimal place to 8. Choose whatever looks good to you.
`   attach("C:/data/myfuns.rda", pos = 2)`	Puts **`C:\r\data\myfuns.rda`** on the search path in position 2. Note that a "\" in Windows corresponds to a "/" in R.
`    attach("data.rda", pos=2)`	Looks in the startup directory for a file called **`data.rda`**. If it finds it, it attaches it in position 2 (**`myfuns.rda`** gets bumped to position 3).

`#   load packages that you often use, e.g.,`
`#   library( "survival" )`
	This command places the survival package on the search path. The survival package contains many functions that are used in survival analysis. Obviously, one only does this is one plans to use these functions a lot.

`    }`	Ends the function definition.

The **`.First`** function will configure your version of R whenever you start R, *provided that* **`.First`** *is in a file on the search path during the startup process*. To make this happen, we first need to put **`.First`** into a file, and then make sure that this file is always on the search path during startup.

Step 2. Save **`.First`** to a file names "myfuns.rda".

Table 11. Save the **`.First`** function to a file.

# R Code	# Explanation
`NameofFunctionFile = "myfuns.rda"`	Choose a name for your function file. I have chosen the name "myfuns.rda", but any other name will do.

`# check that NameofFunctionFile does not already exist.`
`(f.exists = file.exists( NameofFunctionFile ) )`
`if (f.exists) {`
 	
`   warning("A file named ",`
`      NameofFunctionFile,`
`      " already exists.  Either change the name \n",`
`      "of the function file or allow it to be ",`
`      "replaced with a new file." )`
 	
`   go.ahead = FALSE`	

# R Code	# Explanation

`} else {      #end 'if (f.exists)'`
`   go.ahead = TRUE`
`}  #end 'else' of 'if (f.exists)'`	

`go.ahead.anyway =  ( go.ahead | FALSE )`	If you want to replace an existing file named "myfuns.rda", then set **`go.ahead.anyway`** to **`TRUE`**. At present, it is set to be **`FALSE`** if "myfuns.rda' already exists.
`save( .First, file = NameofFunctionFile )`	Saves **`.First`** to the file named by **`NameofFunctionFile`**.

Step 3. Alter **`.Rprofile`** or **`Rprofile.site`** so that "myfuns.rda" is always on the search path during startup.

I assume that the **`.First`** function is in a file **`C:\r\data\myfuns.rda`**. If it is in a different file, you will have to make appropriate changes to the following instructions. At this point you have a choice:

Option 1: Create a text file called **`.Rprofile`** and put it in your user's home directory. See Table 9 for the explanation of how to find the user's home directory. Place the single R command: **`attach( "C:/r/data/myfuns.rda" )`** in **`.Rprofile`** and save **`.Rprofile`** to the user's home directory.

* Note that there is a period at the beginning of the name "**`.Rprofile`**".
* Also, note that **`.Rprofile`** must be a text file with no file extension - it cannot be a Word document file like **`.Rprofile.doc`**.
* If there already exists a file named **`.Rprofile`** on your computer, you can edit it and add the command, **`attach( "C:/r/data/myfuns.rda" )`** to this file.

Option 2: Edit the file **`Rprofile.site`** in in the **`...\etc\`** directory of your hard drive. Again, see Table 9 for an explanation of how to find this directory. Place the single R command: **`attach( "C:/r/data/myfuns.rda" )`** in **`Rprofile.site`** and save this file.

* Note that there is no period at the beginning of the name "**`Rprofile.site`**".

You should use either Option 1 or Option 2, but not both. If you use Option 1, then the **`.First`** function in **`myfuns.rda`** will control the startup when you are logged onto your computer, but if someone else is logged onto your computer under a different user name, this **`.First`** function will be ignored. If you use Option 2, then the **`.First`** function in **`myfuns.rda`** will control the startup when any user runs R on your computer, regardless of the user's logon id.

Now you are done with your R configuration. Quit R, and then restart R. Give the command: **`search()`**. You should see that **`file:C:/r/data/myfuns.rda`** is on the search path. To check that the **`.First`** function has run, give the command: **`options( "continue" )`**. If **`.First`** has run, you will see:

```
$continue
[1] "& "
```

whereas if **`.First`** has not run, you will see:

```
$continue
```

```
[1] "+ "
```

The latter is the default continuation character in R. Assuming that the check shows that "myfuns.rda" was indeed placed on the R search path during startup, you are done.

Suppose you later decide that you want to change the startup configuration in R. An easy way to do this is to create a new **`.First`** function, and save it to the file "myfuns.rda" (edit the commands in Tables 10 and 11 to create a new **`.First`** function in a new "myfuns.rda" file.) The **`save`** function in R will let you replace the old "myfuns.rda" file with a new file with the same name. The only problem with this method is that it cause you to destroy the old version of "myfuns.rda" when you create a new one. If "myfuns.rda" were to contain other useful functions, then this would not be a good way to modify the **`.First`** function. The next section suggests a better way to manipulate files on the search path, including a way to revise a **`.First`** function without destroying a pre-existing version of the file that contains it.

All the work below this line is still under revision

At this point, we have a file **`myfuns.rda`** that will always be placed on the search path when R starts up. Next, we create a **`.First`** function, which we will place in **`myfuns.rda`**, that always runs and configures R whenever R starts up.

Step 4. Create the following .First function. The code for this **`.First`** function is almost identical to the code in Table 10 but it has been modified slightly. The modifications will be explained in Table 12.

Table 12

# R Code	# Explanation
`.First = function() {`	Begins the definition of the **`.First`** function.
`options(continue = "& ", digits = 8)`	Sets the continuation character to & and the default number of digits after the decimal place to 8.

`curr.rdefault.packages =`
`    options("defaultPackages")[[1]]`
	Assign the names of the packages that R currently adopts as the default packages to a variable called 'curr.rdefault.packages'.

`for (i in 1:length(curr.rdefault.packages))`
`    library(curr.rdefault.packages[i],`
`    pos=2, character.only=T)`
	This 'for' loop loads these packages onto the search path. Note that we are forcing this to happen here so that we can put the 'data.rda' and 'jm.utilities.rda' files ahead of this position on the search path.

`    # library(bootstrap)`	Issue commands like **`library(bootstrap)`** here. These commands place the associated libraries on the search path.
`attach("C:/r/data/myfuns.rda", pos = 2)`	Puts **`C:\data\myfuns.rda`** on the search path in position 2. Note that a "\" in Windows corresponds to a "/" in R.

if (!any(tolower(list.files()) == "data.rda"))
 { data.doc = paste(getwd(),
    "data.rda contains data objects", sep="")
    save(data.doc, file="data.rda")
	If there exists a file called 'data.rda' in the startup directory, this code has no effect. If no such file exists, this code creates a string variable called 'data.doc' and saves it to 'data.rda' (thereby creating this file). Later you can change the contents of 'data.doc' to give it a more informative

# R Code	# Explanation
`}`	description of 'data.rda'.
`attach("data.rda", pos=2, warn.conflicts=F)`	This command attaches 'data.rda' to the search path.

`Function.Loc = grep("myfuns.rda", search(), ignore.case=TRUE)`
 	
`if (length(Function.Loc) == 2) detach(pos=max(Function.Loc))`
	Because of the procedure that is implemented at Step **2** below, there will be two occurrences of 'jm.utilities.rda' on the search path. This code simply detaches that last occurrence of 'jm.utilities.rda'. The remainder of the code produces an error message if 0 or more than 2 occurrences of 'jm.utilities.rda' are found on the search path.

`}`	Ends the function definition.

Step 5. The only remaining step is the put the **`.First`** function in **`myfuns.rda`**. Once you do this, the **`.First`** function will control the startup procedure for every startup, not just the startups that are initiated in the current directory. Although are different ways to move the **`.First`** function, the easiest is to use the **`move`** function in **`jm.utilities.rda`**. If you have not already done so, download **`jm.utilities.rda`**from http://faculty.washington.edu/jmiyamot/downloads.htm (there are also other places on my website where I keep a copy of this file so you may have downloaded it from, e.g., my course website). Once you have downloaded **`jm.utilities.rda`**, you can proceed as follows:

Table 13

# R Code	# Explanation
`attach("C:/r/data/jm.utilities.rda")`	Attach **`jm.utilities.rda`** to the search path.
`move(.First, to = "myfuns.rda", replace = TRUE)`	Move the **`.First`** function from **`.GlobalEnv`** to **`myfuns.rda`** on the search path, and save **`myfuns.rda`** to the hard drive. The **`replace = TRUE`** clause replaces any existing **`.First`** function in **`myfuns.rda`** with the new version.

You are done! This completes the configuration for Method III.

With this configuration, the **`.First`** function in **`c:\data\myfuns.rda`** will always run at startup no matter what is the startup directory[15]. Additional modifications can be made to the **`.First`** function in **`c:\data\myfuns.rda`** if you decide later to modify the configuration of R. Simply create a new **`.First`** function in **`.GlobalEnv`**, and **`move`** it to **`c:\data\myfuns.rda`**, thereby overwriting the old version of **`.First`**[16].

Because some of the functions in **`jm.utilities.rda`** will be generally useful, I suggest that you place copies of these functions in your **`myfuns.rda`** file. Here is how to do it.

Table 14

# R Code	# Explanation

[15] The only exception to this is that if there exists a **`.First`** in the **`.Rdata`** file of the startup directory, then this function will run at startup and not any other **`.First`** function.

[16] To move a new version of .First to a file that already contains an old version of .First, you need to set replace.object to TRUE, i.e., give the command move(.First, "jmfuns.rda", replace.object = T).

# R Code	# Explanation
`attach("C:/r/data/jm.utilities.rda", pos = 2)`	You can omit this step if **`jm.utilities.rda`** is already attached to the search path.
`move(c("move", "rm.sv", "doc", "attach.jm", "det.jm", "o.type", "rm.all", "save.jm"), from = "jm.utilities.rda", to = "myfuns.rda", copy.only = TRUE, replace = FALSE)`	Copies **`move`**, **`rm.sv`**, **`doc`**, **`attach.jm`**, **`det.jm`**, **`o.type`**, **`rm.all`** and **`save.jm`** from **`jm.utilities.rda`** to **`myfuns.rda`**. Notice that **`jm.utilities.rda`** and **`myfuns.rda`** are designated simply by their names, as opposed to their their names on the path, **`file:C:/r/data/jm.utilities.rda`** and **`file:C:/r/data/myfuns.rda`**, because **`move`** makes use of partial matching of names. The **`replace = FALSE`** clause makes sure that you will not replace any existing objects that have these same names.

The purpose of **`move`** and **`rm.sv`** have been briefly discussed above. To get more information about these functions, run the R commands: **`doc(move)`**, **`doc(rm.sv)`**, **`doc(doc)`**, etc.

8. A search path with a convenient organization TOC

This section can be skipped if you are a novice user of R. It is primarily intended for people who have decided to use R frequently for data analysis and modeling.

If the **`.First`** function shown in Table 12 configures the R startup, then the initial search path will look like:

```
[1] ".GlobalEnv"        "file:data.rda"       "file:C:/r/data/myfuns.rda" "package:stats"
[5] "package:graphics"  "package:grDevices"   "package:utils"           "package:datasets"
[9] "package:methods"   "Autoloads"           "package:base"
```

Organizing the search path in this way, makes it easier to keep track of the R functions and objects that one is using in an analysis. In any analysis, one creates many functions that are temporarily useful but have no lasting value. In the long run, it is confusing to have these functions mixed in with other functions that have permanent value. Furthermore, one often creates data objects that are temporarily useful, but also have no lasting value. Again, it is confusing to have these data objects mixed in with objects that have permanent value. To maintain a clear distinction between functions and objects that have temporary value from functions and objects that have permanent value (to me), I keep data objects that have permanent value in a file called **`data.rda`**, and I keep functions that are useful when working on any project in **`C:/r/data/myfuns.rda`**.

Of course, to work with the search path shown in avove, I need a function that moves new objects or functions from **`.GlobalEnv`** to **`data.rda`** or to **`C:/r/data/myfuns.rda`**, and a function that deletes objects in **`data.rda`** and **`C:/r/data/myfuns.rda`** if and when I decide they are no longer useful. These functions (**`move`** and **`rm.sv`**) can be placed in your **`myfuns.rda`** by using the code in Table 14. When I create new functions or data objects, they are initially stored in **`.GlobalEnv`**. If I want to save data objects permanently, I move them from **`.GlobalEnv`** to **`data.rda`**. If I want to save functions permanently, I move them from **`.GlobalEnv`** to **`C:/r/data/myfuns.rda`**. By keeping objects and functions that have permanent value in files that are separate from **`.GlobalEnv`**, I avoid the mistake of accidentally deleting them or overwriting them. Furthermore, because my default configuration always has **`C:/r/data/myfuns.rda`** on the search path, the functions that I have created for my own needs can be accessed when working on any project that is located in any startup directory. Finally, let me point out that if you don't follow this suggested procedure and instead keep all functions and objects in a single **`.GlobalEnv`**, you will soon find that you have so many functions and objects in **`.GlobalEnv`** that you cannot easily find them or keep track of what they are for.

9. A Note on the Usefulness of Programming Editors

It is easier to use R if you use a programming editor while writing the code. The following are a few of the many good programming editors that are written for the Windows operating system.

R Script Editor	Run R. On the File menu, you can start a new script or open an existing script. Ctrl-R runs the current selection in the script editor.
Rstudio	★ Free download from http://rstudio.org/. Runs on Windows, Mac and Linux. It is specially designed for R programming and is probably the best choice at the moment.
Tinn-R	Tinn-R is a free programming editor that is designed for working with R. It is available at http://sourceforge.net/projects/tinn-r/.
Notepad++	Notepad++ is a free text editor. It has many features that are designed to help write computer code. Get it at http://notepad-plus.sourceforge.net/uk/site.htm.
Crimson	Crimson is a nice, free text editor. Get it at http://www.crimsoneditor.com/.
Emacs & Ess	Emacs is a well known programming editor. ESS is a version of Emacs that is tailor made for use with R. You can download the software for free from http://www.usc.edu/isd/doc/statistics/help/multiuse/ESS.shtml or http://www.stat.math.ethz.ch/ESS/.
Ultraedit	Go to http://www.idmcomp.com/ for information about this programming editor. Click on Downloads to download a free trial version of this software. You have to pay $30 if you continue to use it for more than 45 days.

10. Downloading R Documentation

Instructions for Downloading R Manuals.

The manuals will be saved to your hard drive as Acrobat pdf files. Put them in whatever directory will be convenient for you.

1. Go to http://www.r-project.org/. Click on Manuals.
2. Click on *An Introduction to R*. This will automatically download an R reference manual.
3. Click on Contributed.
4. Download: ``*Using R for Data Analysis and Graphics*'' by John Maindonald (PDF [695kB], and the data sets and scripts that are available at JM's homepage.
5. Download: ``*Notes on the use of R for psychology experiments and questionnaires*'' by Jonathan Baron and Yuelin Li, and the ``*R reference card*'' by Jonathan Baron.

R-FAQs:

Go to http://www.ci.tuwien.ac.at/~hornik/R/R-FAQ.html.

Documentation for Using R

The R-program comes with documentation through its help system. You can also get free documentation from several users of R. Go to http://cran.r-project.org/. Under the Documentation headings, click on Contributed. You will see the following list of contributed documentation.

``Using R for Data Analysis and Graphics'' by John Maindonald (PDF [702kB], data sets and scripts are available at John Maindonald's homepage (http://room.anu.edu.au/~johnm/).

"Simple R" by John Verzani. Good elementary introduction. Free.

``R for Beginners / R pour les débutants'' by Emmanuel Paradis, an introduction in English (PDF [152kB]) and French (PDF [280kB]).

"Practical Regression and Anova using R" by Julian Faraway (PDF [1MB], data sets and scripts are available at the book homepage). This is a concise yet reasonably comprehensive description of regression and anova computations in R.

``Kickstarting R (version 1.2)'' compiled by Jim Lemon, a short introduction in English as HTML files: donload as gzipped TAR [64kB] or ZIP [81kB]; or browse directly.

``Notes on the use of R for psychology experiments and questionnaires'' by Jonathan Baron and Yuelin Li (HTML [116kB], PDF [235kB]).

``R reference card'' by Jonathan Baron (PDF [58kB], LaTeX source [5kB]).

"Notes on the use of R for psychology experiments and questionnaires" by Jonathan Baron and Yuelin Li. Nice summary.

``Einführung in S'' by Günther Sawitzki (PDF [884kB]), lecture notes (in German) for a 4-5 day introductory course in programming in the S language for students with basic knowledge in probability theory. See also the StatLab Heidelberg S page for more information.

A Spanish translation of **``An Introduction to R''** by Andrés González and Silvia González (PDF file [660kB], Texinfo sources)

I find R for Beginners, Baron's Notes on the Use of R for Psychology Experiments, and the R Reference Card to be especially helpful. Maindonald's, Faraway's and Lemon's notes are also useful.

11. Appendix I: Examples that use the functions[17]: `attach.jm`, `doc`, `move`, `o.type`, `rm.sv`, `setwd.jm`

Table 15. Examples: ***`attach.jm`***

# R Code	# Explanation
`getwd()`	Displays the current working directory for this R session.
`list.files()`	Displays all files in the current working directory. Check to make sure that there is a file called 'data.rda' in the current working directory. The following example assumes that it exists.
`search()`	Displays the current search path.

[17] These functions are all in **`jmfuns.rda`**. Instructions for downloading **`jmfuns.rda`** are given in Section 6 in the discussion of the **`move`** function.

Table 16. This code contrasts the standard **`attach`** function with **`attach.jm`**.

# R Code	# Explanation

`attach("data.rda")`
 	
search()
	The **`search`** command should show 2 copies of **`data.rda`** on the search path. The standard **`attach`** function attaches 2 copies of **`data.rda`** to the search path.

`attach.jm("data.rda")`	This command had no effect because **`data.rda`** was already on the search path.
`attach.jm("data.rda", refresh.path = TRUE)`	If 'refresh.path = TRUE', & 'filename' is already attached to the search path, then all older versions of 'filename' are detached from the search path, and a new 'fresh' version is attached to the search path.

The main advantages of **`attach.jm`** over **`attach`** are:

- If you use **`attach`** inside of a **`for`** loop, you can accidentally attach a large number of copies of a file to the search path. With **`attach.jm`**, this won't happen.
- **`attach.jm`** gives explicit feedback regarding what the **`attach`** command has done to the search path.

Examples: **`doc`**

The **`doc`** function assumes that there are objects **`X`** and **`X.doc`** on the search path. The command **`doc(X)`** prints **`X.doc`** to the screen with a (more or less) attractive formatting.

Table 17. **`doc`** function

# R Code	# Explanation

`xx = 11:15`
`xx`
	Create a vector **`xx`** that contains the numbers 11 through 15.

`xx.doc = "`
`This is sample documentation for the object 'xx'.  The rest of this text is just verbiage to show how 'doc' handles long character vectors. Blah, blah, blah, blah, blah, blah, blah, blah, blah, blah, ....`
 	
`This is paragraph 2 of the documentation.  blah, blah, blah, blah, blah, blah, blah, blah, blah, ....`
`"`
	Create a character vector that contains documentation.

`xx.doc`	Note that the display of **`xx.doc`** is not very pretty.
`doc(xx)`	The **`doc`** function looks for an object called **`xx.doc`** that corresponds to **`xx`**. When it finds it, **`doc`** displays **`xx.doc`** in a more or less attractive way. Note that **`doc`** also tells you what type of object **`xx`** is.
`doc(move)`	Displays the documentation for **`move`** that JM created when he wrote **`move`**.
`doc(attach.jm)`	Displays the documentation for **`attach.jm`** that was entered with the R code for **`attach.jm`**.
`doc(rm.sv)`	(mutatis mutandis)

# R Code	# Explanation
`doc(o.type)`	(mutatis mutandis)
`doc(setwd.jm)`	(mutatis mutandis)

Examples: **`move`**

The **`move`** function moves objects from one file on the search path to another file on the search path. The file on the search path to which the objects are to be moved can be specified either by **`pos=k`** where k = the position number of the destination on the search path, or as an environment name, as in **`env.on.path = "jm.utilities.rda"`**. In the latter case, the **`move`** function finds the unique file on the search path that contains **`"jm.utilities.rda"`** in its name. You do not have to specify the full name of an environment - all that matters is that you specify a character string that uniquely identifies an environment, e.g., **`env = 'jmf'`** and even **`'jmf'`** by itself will specify the **`jm.utilities.rda`** file on the search path provided that it is the only file on the search path that contains "jmf". An error message is generated if both **`pos`** and **`env.on.path`** are specified and if they specify conflicting environments.

Table 18. Examples: **`move`**. These examples assume that **`data.rda`** is on the search path.

# R Code	# Explanation

`xx = 11:15`
`xx.doc = "`
`This is sample documentation for 'xx'.  Blah, blah,`
`blah, blah, blah, blah, blah, ....`
`"`
	Create a vector **`xx`** that contains the numbers 11 through 15. Create a character vector that contains documentation. If **`xx`** and **`xx.doc`** already exist, you don't have to run this command.

`doc(xx)`	Display the documentation for **`xx`**.

`yy = matrix(1:12, ncol=3, byrow=T)`
`dimnames(yy) = list(`
`   c('A','B','C','D'), c('P','Q','R'))`
 	
`yy.doc = "This is sample documentation for 'yy'."`
	Create a 4 x 3 matrix **`yy`** and documentation for **`yy`**.

`ls()`	**`ls`** displays the objects that are in **`.GlobalEnv`**, the top-most file on the search path. Note that **`xx`**, **`xx.doc`**, **`yy`**, and **`yy.doc`** are in **`.GlobalEnv`**.
`search()`	Check that **`data.rda`** is on the search path. If it isn't, use **`attach`** or **`attach.jm`** to attach it to the search path.
`move(xx, 'data.rda')`	This command moves **`xx`** to **`data.rda`** and deletes it from **`.GlobalEnv`**.

`ls()`
 	
`ls(pos=2)`
	The first **`ls`** shows that **`xx`** and **`xx.doc`** have both been removed from **`.GlobalEnv`**. The second **`ls`** shows that **`xx`** and **`xx.doc`** are now in **`data.rda`** (assuming that **`data.rda`** is in position 2 on the search path).

`move(yy, pos=2)`	Assuming that **`data.rda`** is in position 2 on the search path, this command will move **`yy`** and **`yy.doc`** to position 2.

Table 19. The next code illustrates what happens if you try to move an object to a file that already contains an object with the same name.

# R Code	# Explanation

`search()`
`ls(2)`
	Check that **`data.rda`** is on the search path in position 2, and that **`xx`**, **`xx.doc`**, **`yy`**, and **`yy.doc`** are all in **`data.rda`**.

`xx = 3.14159`
`xx.doc = "The new 'xx' has the same value as pi."`
	Create a new **`xx`** and **`xx.doc`**. These objects are in **`.GlobalEnv`**.

`move(xx, 'data.rda')`	Assuming that **`data.rda`** already contains objects named **`xx`** and **`xx.doc`**, this command should have no effect other than to produce a warning.
`move(xx, 'data.rda', rep = T)`	**`rep = T`** is an abbreviation for **`replace.objects = TRUE`**. By setting **`rep = T`**, we cause **`move`** to overwrite the existing versions of **`xx`** and **`xx.doc`** in **`data.rda`**, with the new versions that were created in **`.GlobalEnv`**.

`ls()`
`find('xx')`
`find('xx.doc')`
	These commands show that **`xx`** and **`xx.doc`** are now in **`data.rda`**.

Table 20. The following example shows how to move multiple objects.

# R Code	# Explanation

`xx = 12345`
`yy = 54321`
	New objects named **`xx`** and **`yy`**.

`move(c("xx", "yy"), "data.rda")`	Tries to move both **`xx`** and **`yy`** except that **`data.rda`** already contains objects by those names.
`move(c("xx", "yy"), "data.rda", rep = T)`	This succeeds in moving both **`xx`** and **`yy`** to **`data.rda`**. Note that the **`move`** function always tries to move **`.doc`** objects, but it executes the move even if it can't find corresponding **`doc`** objects.

Table 21. The following example shows that care must be taken when moving a character vector.

# R Code	# Explanation
`nn = c("Bob", "Mary", "Jim", "Harry")`	Create a character vector.

`ls()`
 	
`ls('file:data.rda')`
	Note that **`nn`** exists in **`.GlobalEnv`** but not in **`data.rda`**.

`move(nn, 'data.rda')`	This command does not move **`nn`** to **`data.rda`**. This command thinks that you want to move objects named **`Bob`**, **`Mary`**, **`Jim`**, and **`Harry`** to **`data.rda`**, and it can't find these objects.
`move("nn", 'data.rda')`	This command works because it is clear that you want to move **`nn`** and not the objects named by the character components of **`nn`**.
`ls()`	These commands show that the desired move was successful.

# R Code	# Explanation
`ls('file:data.rda')`	

Example: ***o.type***

The **`o.type`** function displays information about the type of object that a given object is. For example, **`o.type( X )`** displays information about the type of object that **`X`** is.

Table 22.

# R Code	# Explanation

`aa = 5`
`o.type(aa)`
	Shows that **`aa`** is a scalar (single number). In R terminology, a scalar is also a vector of length 1. The **`o.type`** function deviates from this terminology and calls a vector that is of length 1 a scalar but not a vector.

`vv = c(3,5,2,5,3,7,2)`
`vv`
`o.type(vv)`
	Shows that **`vv`** is a numeric vector.

`LL = list(aa = aa, vv = vv)`
`LL`
`o.type(LL)`
	`LL` is a list. R counts lists as a type of vector. **`o.type`** labels lists as non-vectors, i.e., **`o.type`** deviates from R terminology.

`hh = c("A", "AB", "AC", "AD")`
`hh`
`o.type(hh)`
	Character vector.

`v1 = c(3,5,2,8)`
`v2 = c(6,3,1,6)`
`subj = c("Bob", "Jim", "Bill","Mary")`
`cond = c("A","A","B","B")`
`dframe = data.frame(subject = I(subj),`
`    condition = cond, v1 = v1, v2 = v2)`
`dframe`
`o.type(dframe)`
	A dataframe is a kind of list. It is analogous to an SPSS data set. So **`o.type`** classifies **`dframe`** as being both a list and a dataframe. The notation **`I(subj)`** is needed to prevent R from turning the **`subject`** variable into a factor (don't worry about this now if you don't understand this).

`o.type(dframe, var = T)`	**`var = T`** abbreviates **`variables = TRUE`**. If the object is a dataframe, **`var = T`** causes **`o.type`** to display the object type of each variable in the dataframe.

Example: **rm.sv**

The **`rm.sv`** function deletes (removes) an object from a file on the search path and then saves that file.

Table 23.

# R Code	# Explanation

`search()`
`ls("file:data.rda")`
`ls(2)`
	These commands are just two different ways to check the contents of **`data.rda`**. The second version, **`ls(2)`**, assumes that **`data.rda`** is in the second position on the search path. The following examples assume that there are objects, **`nn`**, **`xx`**, **`xx.doc`**, **`yy`**, and **`yy.doc`**, in **`data.rda`**.

`rm.sv('xx', 'data.rda')`
`ls(2)`
	You should see that **`xx`** and **`xx.doc`** have been removed from **`data.rda`**. Note that **`rm.sv`** has also saved **`data.rda`** so that **`xx`** and **`xx.doc`** will NOT be present in this file the next time it is loaded onto the search path of a different R session.

`rm.sv(c("nn", "yy"), pos=2)`	This comand shows that you can remove and save multiple objects from a file on

# R Code	# Explanation
	the search path.

Example: **`setwd.jm`**

The **`setwd.jm`** function sets the working directory for R. This example assumes that you have two directories, **`c:\AA\data`** and **`c:\BB\work`** on your computer. You can change the names to anything else that works on your computer.

Table 24. **`setwd.jm`**

# R Code	# Explanation

`setwd.jm("c:/AA/data")`
`getwd()`
`search()`
	`setwd.jm` creates and attaches a file **`data.rda`** in **`c:\AA\data`**. The **`getwd`** command verifies that **`c:\AA\data`** is the current working directory.

`find("ThisFile.doc")`
`doc(ThisFile)`
	Note that an object called **`ThisFile`** is automatically placed in **`data.rda`**.

`ThisFile.doc = "`
`This is better documentation for c:\\AA\\data. Blah, blah, blah, blah, blah, blah, blah, ..."`
 	
`move("ThisFile.doc", 'data.rda', rep=T)`
`doc(ThisFile)`
	The default documentation in **`ThisFile.doc`** can usually be improved.

`aa = 3`
`bb = 4`
	Create two object, **`aa`** and **`bb`**, in **`.GlobalEnv`**.

`setwd.jm("c:/BB/work")`
`getwd()`
`search()`
`doc(ThisFile)`
	Note that **`setwd.jm`** has changed the working directory to **`c:\BB\work`**. You can tell that this is true from the output of **`getwd`**, **`search`**, and **`doc(ThisFile)`** (remember that **`ThisFile`** and **`ThisFile.doc`** were created and placed in **`c:\BB\work\data.rda`**.)

The effect of **`setwd`** and also of **`setwd.jm`** is to change the directory (folder) that R thinks is the current working directory. When you give a command like **`attach("Useful.data")`**, R expects to find a file called **`Useful.data`** in the current working directory, so you need to be able to control what is the current working directory. Also, in my system, the **`data.rda`** file is the set of data objects for a particular project, so when I change the current working directory, I change the **`data.rda`** file to the file for a different project.

12. Appendix II. Transferring data to and from SPSS TOC

To illustrate the transfer of data from SPSS to R, we will need a data set in the SPSS format. We will use the NewDrug.sav data set:

- Go to: http://people.cst.cmich.edu/lee1c/spss/Prjs_DataSets.htm
- Search for the "**`New Drug.sav`**" (note that there is a space between "**`w`**" and "**`D`**"). Right click on this link and download the file. It will be downloaded as a file named, "**`NewDrug.sav`**".
- If you have SPSS on your computer, you can open this file to verify that it is indeed an SPSS data file.

The following example assumes that **`NewDrug.sav`** has been saved to the folder, **`C:\data`**.

Table 25.

# R Code	# Explanation
`library( foreign )`	The **`library`** function puts a package of functions on the search path. Note: You must have previously downloaded the package, e.g., by giving the command: **`install.packages`**(<package name> **)**

`ND = read.spss(`
`file = "C:/r/data/newdrug.sav",`
`use.value.labels = TRUE,`
`to.data.frame = TRUE )`
	`read.spss` is a function in the **`foreign`** package - it reads data into R from SPSS .sav files. **`file`** is the path and name of the file. If **`use.value.labels`** is **`TRUE`**, the SPSS value labels are read into R as the level names of a factor. If **`use.value.labels`** = **`FALSE`**, then the numeric codes for the levels are read into R. If **`to.data.frame = TRUE`**, the output of **`read.spss`** is a dataframe. If **`to.data.frame = FALSE`**, the output of **`read.spss`** is a list that is not a dataframe.

To see how to send data back from R to SPSS, we will use the **`CO2`** dataset as an example. The **`CO2`** dataset is automatically included in the installed version of R; it is in the **`datasets`** package. In a typical R installation, the **`datasets`** package is automatically placed on the search path when R starts up. To check whether you have this dataset on your computer, run the commands:

Table 26. Look for the **`CO2`** dataset.

# R Code	# Explanation
`search()`	You should see **`package:datasets`** on the search path.

`# If you do not see "package:datasets" on`
`# the search path, run the command:`
 	
`library(datasets)`
	This should not be necessary because **`package:datasets`** should already be on the search path, but if it is not, run the command to the left. If this command gives an error message, then go to the next command.

`# If the preceding call to 'library' fails`
`# to load the datasets package, run:`
 	
`install.packages("datasets")`
	This should not be necessary because the **`datasets`** package should already be installed on your computer, but if it is not, then the command to the left should install **`datasets`** on your computer.

Assuming that the **`datasets`** package is indeed on your computer and that **`package:datasets`** is on the search path, the following will let you look at the **`CO2`** dataset.

Table 27.

# R Code	# Explanation
`CO2`	Although this shows you the entire **`CO2`** dataset, it is too large to view comfortably on the screen.
`CO2[ 1:15, ]`	**`CO2`** has 5 variables.
`o.type( CO2, variables = TRUE )`	**`CO2`** is a dataframe that belongs to a number of classes. The **`Plant`**, **`Type`**, and **`Treatment`** variables are factors; the **`conc`** and **`uptake`** variables are numeric.
`?CO2`	See the documentation of this dataset.

Table 28. Now we will transfer these data to SPSS. This code assumes that **`jm.utilities.rda`** is on the search path.

# R Code	# Explanation
`export.spss( CO2, outfile = "C:/r/data/co2" )`	This code creates two files in C:\data. **`co2.txt`**: This file contains the data from the **`CO2`** dataframe (in R). These data are separated as a space delimited ASCII file. **`co2.sps`**: An SPSS syntax file that contains SPSS commands that load the **`CO2`** data and

Now:

- Run SPSS
- Open the syntax file, **`co2.sps`**.
- Run the SPSS syntax in **`co2.sps`**.

The **`CO2`** dataset should now be in SPSS.

End of Rnote Document