SAVING AND LOADING OBJECTS IN R

The Workspace and History

You've probably noticed already, when you start and quit R, you see something like this...

```
R version 2.6.2 (2008-02-08)
Copyright (C) 2008 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

.
.
.

[Workspace restored from /Users/billking/.RData]

> quit()
Save workspace image? [y/n/c]:
```

Note: In Windows you get a pop-up window when you quit.

Your workspace, remember, consists of all the data objects you've created or loaded during your R session (and perhaps a few other things as well). When R asks if you want to save your workspace image before quitting, and you say "no", all of the NEW stuff you've created goes away--forever. If you say "yes", then a file called ".RData" is written to your working directory. The next time you start R *in this directory*, that workspace and all its data objects will be restored.

This is extremely convenient for people who are running R in a terminal, such as the terminal.app in OS X or a Linux terminal. Those people can devote a working directory to each problem they are working on, and when they start R, they can simply change to that directory FIRST (before starting R), and R will open with that directory as the working directory and restore the workspace from there. It's not so convenient for people who are running R in Windows or in the Mac R GUI console, in which case R will always start in the same default working directory. In Windows XP that will be "My Documents". In Windows Vista and (I suspect) Windows 7, it is the user's home directory. On the Mac and in Linux it will be the user's home directory. (If you don't know what your home directory is, then start R and use `getwd( )`. Then you'll know!)

There is, of course, a work-around. I'll describe how it would work in Windows XP. Mac users, I suspect, will be smart enough to see the immediate generalization.

In Windows XP, R opens by default in "My Documents". This default behavior can be changed, but there is little point in doing so. When R opens, it will load the workspace file (".RData") if there is one, and it will also load the history file (".Rhistory") if there is one. (The history file is a list of commands entered during the previous R session.) If you then change your working directory, to "Rspace" let's say, R will bring along the already loaded workspace and history files, but it WILL NOT load any such files you've stored in "Rspace".

Let's say you have the following work habits. You open R, you change to "Rspace", you do your R business, you quit R and opt to save the workspace. The next time you start R, your previously created data objects are nowhere to be seen, even after you change the working directory to "Rspace". Here's how to retrieve them. First, you may want to get rid of any workspace items R has dragged along from the default working directory. That can be done with the `rm( )` function. Then you can load the previously saved workspace (and presumably the history file if so desired) like this...

```
> ls()                                  # This is my default workspace from My Documents.
[1] "age.out"       "m.ex"          "m.ex.minussex" "outcome.out"
[5] "respiratory"   "seizure"       "visit.matrix"
> setwd("Rspace")                       # First change, if you haven't already.
> rm(list=ls())                         # Delete the default workspace.
> load(".RData")                        # Load a previously saved workspace.
> loadhistory()                         # Load a previously saved history file.
> ls()
[1] "my.data"
```

You can see that I got .RData from Rspace, because it contains the "my.data" data frame that we created in the previous tutorial. There are also menu entries in the R GUI for loading and saving the workspace and history. In the Windows GUI they are under the File menu. On the Mac the menu items for the workspace are under the Workspace menu. I don't see anything for working with the history file.

If you change the workspace, say by removing "my.data", but then don't save it when you quit or before you change the working directory, "my.data" will still be there when you come back next time. The `rm( )` function modifies the workspace, but NOT the workspace image file (".RData"). If you remove "my.data" and save the workspace when you quit, "my.data" will be gone because the new workspace image (.RData file) will overwrite the old one. If you change working directories and then load ".RData" in the new directory, R will ADD it to whatever you've brought with you from the previous working directory. If you don't want that, be sure to clear the old workspace before loading the new one.

If your work sessions are long, it's a good idea to save a workspace image once in awhile, because this image is held in RAM. That means if the power goes out, it's gone. You can save a workspace image at any time by doing this...

```
> save.image()                    # Save an .RData file.
> savehistory()                   # Save an .Rhistory file.
```

You can save multiple workspaces in the same directory by specifying a file name as an option to the above command...

```
> save.image(file="workspace21.RData")
```

By the way, I should mention that the dot in front of the name ".RData" makes the file invisible (or hidden) in Linux and Mac OS X directories, so you Maccies will not see it in the Finder. It has no such effect on Windows systems. I should also tell Windows users, who are not used to this, that capitalization is important when loading the workspace image file. It must be ".RData". The same goes for loading other files in the exercises below.

It takes awhile to get used to how the workspace and history files work, when they are saved, when the existing ones are modified, and so on, but it's really quite logical. R is a good old fashioned command line program. It does not do anything that you don't tell it to do. This is one of its BEST features as far as I'm concerned! If you want it to do things automatically upon startup and shutdown, there are scripts you can modify, but that is beyond the scope of this tutorial.

Scripts

In addition to working interactively with R at the command line, R can also be scripted. A script is just a text file of R commands (without the command prompts). For example...

```
# A comment: this is a sample script.
y=c(12,15,28,17,18)
x=c(22,39,50,25,18)
mean(y)
mean(x)
plot(x,y)
```

This script will create two vectors, y and x. It will find the mean of each. Then it will plot a scatterplot. Be careful. If you read this into an R session, it will execute those commands just as if you'd typed them at a command line! Anything in your workspace called x and y will be overwritten.

While R has a built-in script editor, I find it handier to use my own text editor, like Notepad (ugh!), vim, or TextWrangler. (The script editor in the Windows version of R is quite good. Pull down the File menu and choose New Script to open it. On the Mac I don't see any reference to a script editor in the menus, and in fact there doesn't appear to be one. If you are running in a terminal, there won't be a script editor. You can copy and paste scripts from any text editor, however.) In this case, I've included this script as its own page at this website. Here's what to do. Click on the link below, right click on the page that appears and choose "Save Page As...", and save the page as a text file to your working directory. (Or save it to the Desktop and move it to your working directory.)

Here is the link to the sample script.

Once the script is saved in your working directory, simply type...

```
> source("sample_script.txt")
```

Or if you have an adventurous streak, you could try to source in this script directly from the Internet...

```
> source(file="http://ww2.coastal.edu/kingw/statistics/R-tutorials/text/sample_script.txt")
```

When you run this script, you will see the graph, but you will not see the means. Why not? Because you didn't ask for them. Generally in a script, only the last line returns output. If you want other lines to do so, you have to use the explicit `print( )` function...

```
# A comment: this is a sample script.
y=c(12,15,28,17,18)
x=c(22,39,50,25,18)
```

```
print(mean(y))
print(mean(x))
plot(x,y)
```

That's all there is to reading in and executing a script. (You might want to remove the x and y variables it created in your workspace.) Scripts are a good way to save long sequences of commands that you might want to execute over and over again, or just to save procedures that you may not remember from one day to the next. It's also a good way to save the procedure for creating a complex graphic.

Saving and Loading the Easy Way

The easiest way to save and load data objects you create at the command line is by using the cryptically named `save( )` and `load( )` functions. Any data object--a vector, a table, a data frame, the output of a statistical procedure--can be saved to the working directory very simply, as follows...

```
> rm(list=ls())                          # clean up
> y.vector = runif(20)                   # a vector of 20 random numbers
> save(y.vector, file="yvec.saved")      # save to the working directory
> rm(y.vector)                           # removed
> ls()
character(0)
> load("yvec.saved")                     # loaded back in
> ls()
[1] "y.vector"
```

The syntax for `save( )` couldn't be simpler. Tell it the data object you want to save, and then give it a file name in which to save it. The extension on the file name is arbitrary, but I like to use ".saved" so I know that those files are saved data objects. Other common extentions are ".R" and ".rda". The files created by `save( )` are in a binary format (i.e., they are not human readable), so you cannot examine them in a text editor to view their contents. To load the data object at some future time, use the `load( )` function, which only requires that you specify the name of the file. The data object will be placed in your workspace with the same name it had when it was saved.

You don't have to save or load from the working directory either. By specifying a complete path name in the "file=" option, you can put the files anywhere your computer can get to.

```
> rm(y.vector)
```

Reading Files Created Externally

As I mentioned in the last tutorial, the most convenient way to create a data frame is in a spreadsheet program like OpenOffice Calc, iWork Numbers, or Microsoft Excel. I wouldn't have suggested it if there wasn't some way of reading those files into R! R will read files created by a very large number of other applictions, but the easiest way to exchange files with other apps is as plain text files, and that is what I will discuss here. For details on how to read other kinds of files, go to the R-project manuals page and read the "R Data Import/Export" manual:

R Data Import/Export

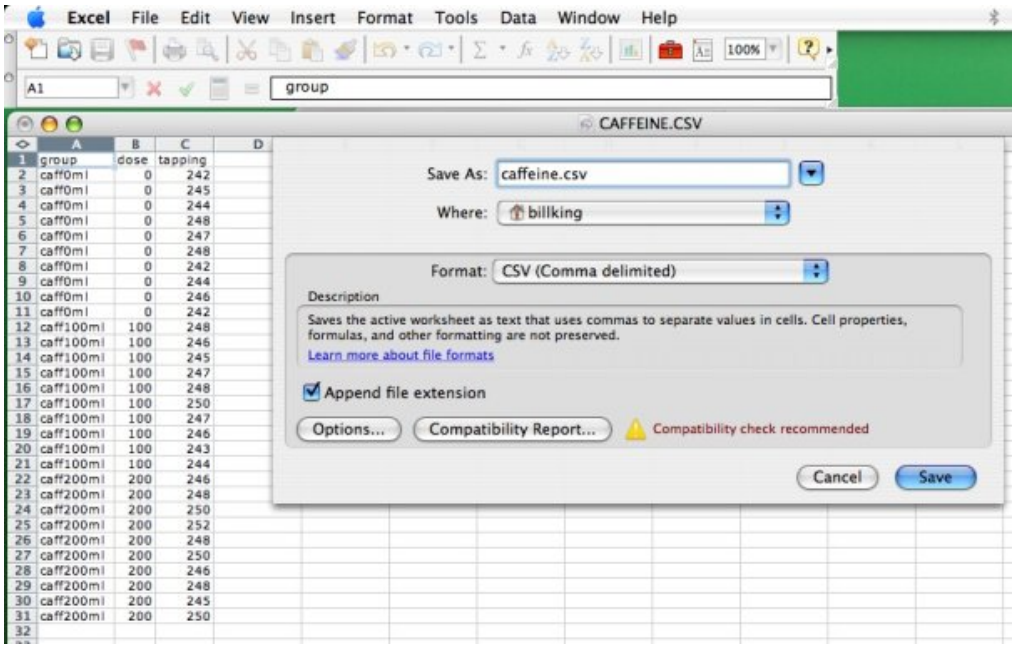
The best way to keep track of data you are collecting and will be analyzing electronically is to type it into a spreadsheet in the form of a data frame. (ATTENTION my Psyc 497 students: THIS IS REQUIRED!) Just about any modern spreadsheet program will do. If you don't have Microsoft Office Excel, you can go to OpenOffice.org and download Open Office for free. Linux fans can try Gnumeric if Open Office is too clunky. For Mac fans, there is also NeoOffice, an Aqua version of OpenOffice.

The following data are from the *Handbook of Small Data Sets* (Hand et al., 1994), and are from an experiment in which caffeine dose is related to a simple motor task--finger tapping rate.

Dose of Caffeine		
0 ml	**100 ml**	**200 ml**
242	248	246
245	246	248
244	245	250
248	247	252
247	248	248
248	250	250
242	247	246
244	246	248
246	243	245

242	244	250

The following figure shows these data entered into an Excel spreadsheet. Notice I have entered three variables: dose as a factor ("group"), dose as a numerical variable ("dose"), and finger tapping rate in taps per minute ("tapping"). Each variable is entered into its own column, and each column has a variable name at the top in a row of headers. There are no blank rows or fancy formatting, just a row of headers and the data values. Period.



	A	B	C	D
1	group	dose	tapping	
2	caff0ml	0	242	
3	caff0ml	0	245	
4	caff0ml	0	244	
5	caff0ml	0	248	
6	caff0ml	0	247	
7	caff0ml	0	248	
8	caff0ml	0	242	
9	caff0ml	0	244	
10	caff0ml	0	246	
11	caff0ml	0	242	
12	caff100ml	100	248	
13	caff100ml	100	246	
14	caff100ml	100	245	
15	caff100ml	100	247	
16	caff100ml	100	248	
17	caff100ml	100	250	
18	caff100ml	100	247	
19	caff100ml	100	246	
20	caff100ml	100	243	
21	caff100ml	100	244	
22	caff200ml	200	246	
23	caff200ml	200	248	
24	caff200ml	200	250	
25	caff200ml	200	252	
26	caff200ml	200	248	
27	caff200ml	200	250	
28	caff200ml	200	246	
29	caff200ml	200	248	
30	caff200ml	200	245	
31	caff200ml	200	250	
32				

At this point, a decision must be made, which is in what form to save the file. I recommend you save it as an Excel spreadsheet first, but R will not easily read it in that form. (It's possible, but not recommended.) So you will also need to save it in plain text form, and the choices are tab separated data values, or comma separated data values.

Each form has its advantages and disadvantages. All things considered and long story short, I prefer the comma separated form, or .csv file. So after you save a copy as an Excel spreadsheet (or whatever program you are using), then save a copy as a .csv file. This is a plain text file that can be examined and modified in a text editor, and which R can read with no problem. (Note: If there are commas in any of your data fields, in a character variable like an address, for example, the csv format will have a problem with this. On the other hand, if there are spaces in any of your data fields, the tab or whitespace separated data format might choke on that. Be careful when you're preparing your data file. Don't use commas, and don't use spaces. R will actually work around both of these problems, but it's just easier to avoid the problem in the first place!)

It wouldn't hurt you to create this file yourself, but you can also download it from this link...

caffeine.csv

Download it and save it in (or move it to) your working directory. Now to get it loaded into R. The appropriate function is `read.csv( )`...

```
> help(read.csv)

Usage
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", row.names, col.names,
           as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = default.stringsAsFactors(),
           encoding = "unknown")

read.csv(file, header = TRUE, sep = ",", quote="\"", dec=".",
         fill = TRUE, comment.char="", ...)
```

Let's look at the help page for this function. The function is a special form of the `read.table( )` function, which will read either type of text file, as well as others with just about any field separator you can come up with, like pipe characters if you work with census data, for example. The argument for the function is the file name. That is followed by an option that tells whether or not there are headers in the file (a row of variable names at the top). The default for `read.csv( )` is "header=TRUE". The next

option tells what the separator is, with the default being a comma. If there are comment lines in the file, set the "comment.char=" option to whatever the comment line character is, usually #. So read the file this way...

```
> read.csv("caffeine.csv") -> caff
```

You can also try reading the file directly from the Internet...

```
> read.csv("http://ww2.coastal.edu/kingw/statistics/R-tutorials/text/caffeine.csv") -> caff
```

This will read in the file and save it in a data frame object called "caff" in your workspace. You can now work with it as you would any other object in your workspace. The `read.csv( )` function can read files from any location your computer can access, including websites and ftp sites on the Internet. All you need to do is supply a complete path name or url in place of the file name. One note: When a data frame is read in using this function, all character variables will by default be read in as factors. Set the "as.is=" option to the names of any variables you do not wish this to happen for (although it really doesn't make much difference).

Okay, so you've worked with the data frame, have done some analyses, and have made some modifications to it. Now you want to write the file back to your working directory as a csv file that is human readable (as opposed to saving in binary format using `save( )` as we did in the previous section, which is also possible). The function is `write.csv( )`...

```
> write.csv(caff, file="caffeine2.csv", row.names=FALSE)
```

Of course, if there are explicit rownames, set the "row.names=" option to TRUE (or T). Another option you can consider setting is "quote=FALSE". This will save the file without quoting the character values. And on the topic of quotes, the file names inside these functions must be quoted. Otherwise, R will consider them to be the names of defined objects and begin looking for their values. This is actually a pretty handy feature, as I'll explain in a future tutorial.

You can also use the `save( )` function to save the "caff" object, but the saved file will not be human readable, and it will not be readable by programs like Excel. Files saved with `write.csv( )` can be read by any program that will read .csv files, including most statistical software (like SPSS) and virtually all spreadsheet programs and text editors.

Saving and Printing the R Console and Graphics Device

The methods for doing this are specific to different operating systems, so pick yours below. So you'll have a graphic to work with, do this...

```
> with(faithful, plot(waiting, eruptions))
```

When you're done with this section, you can close the Graphics Device window just like closing any other window on your system.

Windows

To save a console session: 1) Click in the R Console window to bring it to focus, 2) Pull down the File menu and choose Save to File..., 3) Proceed as you would when saving any other file.

To print a console session: 1) Click in the R Console window to bring it to focus, 2) Pull down the File menu and choose Print..., 3) Be warned that this prints the entire console session, which can be VERY long. If you want to print just a part of it, highlight that part first, then follow steps 1 and 2.

To save a graphic: 1) Click in the Graphics Device window to bring it to focus, 2) Pull down the File menu, choose Save as..., and choose the desired format, 3) Proceed as you would when saving any other file. (Note: If you want to share this graphic with friends who may not be using Windows, DON'T save it as a Metadata file.)

To print a graphic: 1) Click in the Graphics Device window to bring it to focus, 2) Pull down the File menu and choose Print...

Linux

To save or print a console session: There is probably a way to do this, but I have never seen it documented. I highlight what I want to save or print, copy and paste it into a text editor like gedit, and then use that app to save or print.

To save a graphic: In an R terminal session, issue the following command...

```
> dev.print(file="faithful.pdf")
```

You can choose your own file name, of course, and you can also save as a postscript file.

To print a graphic: Proceed as if you were saving (above) but leave out the file name and "file=" option. See ?dev.print for all the details.

Mac OS X

To save a console session: 1) Click in the R Console window to bring it to focus, 2) Pull down the File menu and choose Save As..., 3) Proceed as you would when saving any other file.

To print a console session: 1) Click in the R Console window to bring it to focus, 2) Pull down the File menu and choose Print..., 3) Be warned that this prints the entire console session, which can be VERY long. I don't know that there is a way, from within R, to print just a part of it. I highlight what I want, copy and paste it to a text editor, and go from there.

To save a graphic: 1) Click in the Quartz device window to bring it to focus, 2) Pull down the File menu and choose Save As..., 3) There aren't many options! The file will be saved in pdf format.

To print a graphic: 1) Click in the Quartz device window to bring it to focus, 2) Pull down the File menu and choose Print..., 3) You can also print the image to a pdf file this way.

All Operating Systems

When I say "text editor" in the above notes, I mean text editor, not word processor. If you are copying and pasting from R to a word processor, change the font in the word processor to something like courier new, or some other monospaced (typewriter-like) font. This will keep your tables and so forth aligned properly.

---

revised 2010 August 2

Return to the Table of Contents