Rsession #1: Intro to R, Rstudio, and basic operations

Contents

1 Companion material **1**

2 Getting familiar with Rstudio **1**

2.1 Setting the working directory 3

3 Your first R operations: basic calculator functionality **4**

3.1 Assigning values to variables 5

3.2 Character (text) and logical (TRUE or FALSE) variables. 5

4 Logical variables **5**

4.1 Logical Operators 6

5 If this, do that: the `if` and `if-else` statements **7**

5.1 Formatting: keep your code tidy! 7

6 Saving and reloading your work **8**

6.1 Saving your workspace 8

6.2 Loading an .Rdata file 8

6.3 Functions to do these things the "manual" way 9

1 Companion material

Don't just watch the videos; the exercises are the whole point!

Strongly recommended:

- DataCamp Introduction to R, Chapters 1 (Intro to Basics)
- DataCamp Intermediate R, Chapter 1 (Conditionals and Control Flow)

Recommended, but we'll get back to these topics in more detail later:

- DataCamp Introduction to R, Chapters 3,5,6.

2 Getting familiar with Rstudio

Rstudio is an integrated development environment (IDE) for R, which just means a program that puts all the things you usually do with R (write and save code, output results, make plots, read and save data, etc.) in one nice package with convenient features. Unless you already have a different, preferred workflow for R (in which case you probably don't need to be at this session), we highly recommend you use Rstudio. An open Rstudio session looks something like this:

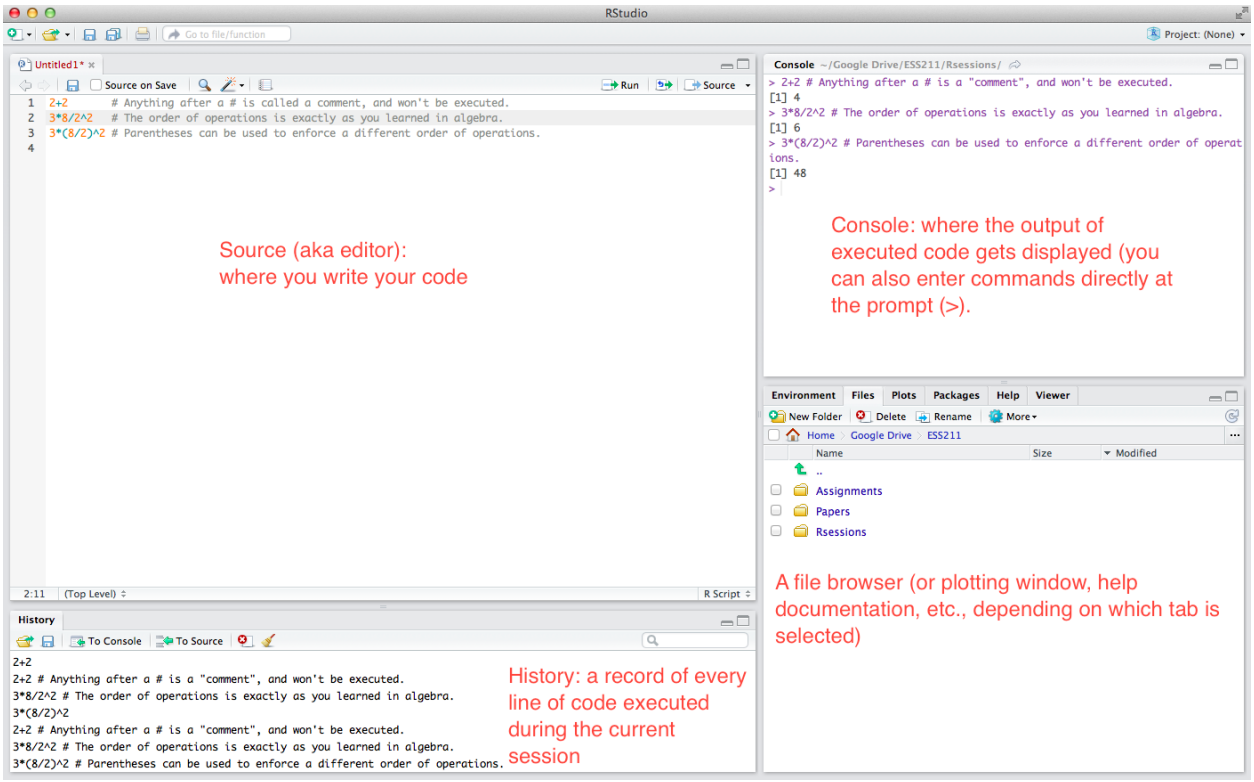


Figure 1: You can drag the borders of each pane around (or minimize or maximize them) to get a configuration you like

Each sub-window is called a pane. Above, you see the Source, Console, History, and Enviroment panes. You can customize which panes go in which quadrant (and which tabs go with which pane) in Tools -> Global Options -> Pane Layout. The above layout was set up with the following:

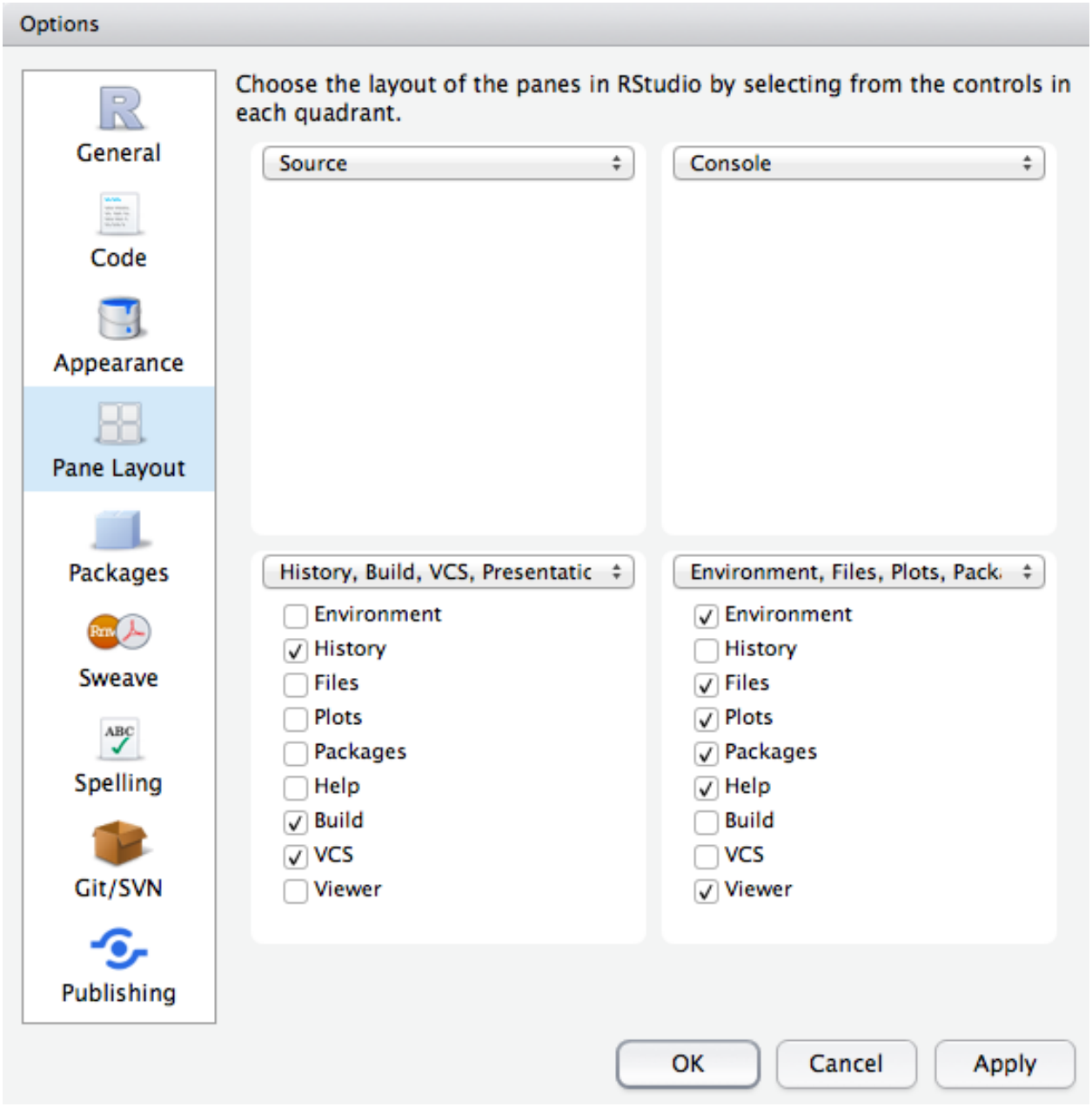


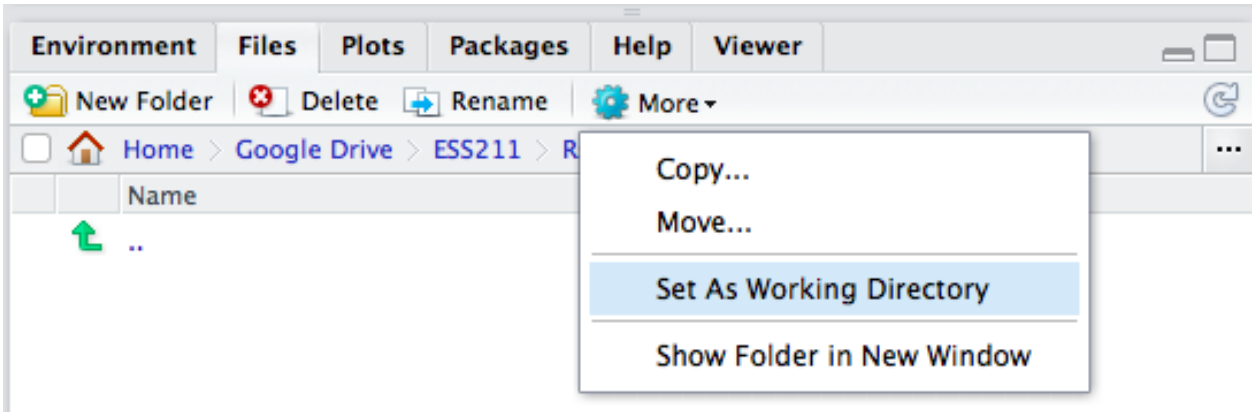
Figure 2: Tools -> Global Options -> Pane Layout

2.1 Setting the working directory

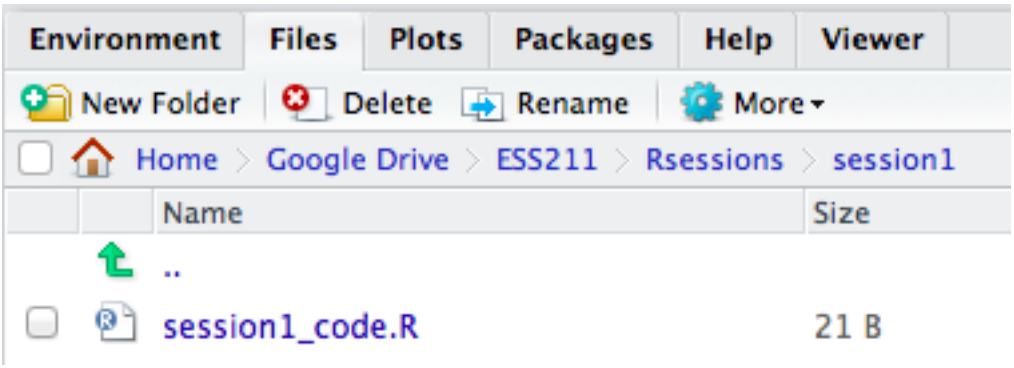
The working directory is the folder where you want any files associated with your current activity (code, figures, input data, etc.) to live. Open the "Files" tab and navigate to the directory where you want to save the work from this session. Use the "New Folder" button to create a new folder, if needed.

Now that you're in your chosen folder, click `More -> Set As Working Directory`. Note the line of code

that automatically executed in the console.



Your source file is probably as-yet empty, and called "Untitled1". Save it with a name of your choice. That file should now appear under Files.



3 Your first R operations: basic calculator functionality

Ok, now we can get on to actually doing something with R. When starting out, it can help to just think of R as a big calculator. You can type a command, like `2+2`, directly at the console prompt and hit `Enter` to execute that command (try it), but you'll want to get in the habit early of writing code in the editor and sending it to the console.

Enter these lines into your editor (we'll use "editor" and "source" interchangeably).

```
2+2 # Anything after a # is a "comment", and won't be executed.
3*8/2^2 # The order of opertions is exactly as you learned in algebra.
3*(8/2)^2 # Parentheses can be used to enforce a different order of operations.
```

Now run these lines in the console two ways:

1) Highlight all three lines, then hit (for a Mac) `cmd+enter` or (for Windows) `cntl+enter`.
2) Without highlighting anything, place your cursor on the first line, then keep hitting `cmd+enter`.

The `cmd+enter` keystroke is a shortcut for the "Run" button in the upper right corner. Mousing up to the "Run" button time and time again gets tedious, and we recommend getting used to the shortcut. As you've

now seen, `cmd+enter` runs either all highlighted text, or, if nothing is highlighted, just whichever line the cursor is on. You can also execute portions of lines. Try highlighting only the `3*8` in the second line, and send it to the console.

We've just used some simple arithmetic operators like the ones you'd find on a calculator, and you can probably guess what they all are.

```
+  addition
-  subtration
*  multiplication
/  division
^  exponentiation
```

These operations become much more useful if we can store their results as variables and perform subsequent operations on those variables, which we'll get to next.

3.1 Assigning values to variables

One very important operator is the "assignment operator",`=`, so called because it assigns the value on its right to the variable on its left (e.g. `x=2`). Thought we're used to reading expressions left to right, it's better to think of assignments as right to left. The right hand side can be an expression like `2+2`, and that expression gets evaluated first. Then, the result of that evaluation gets saved in the variable on the left hand side.

```
> x = 2+2 # evaluates 2+2, and stores ("assigns") the result to the variable "x".
> x # prints the value of x to the console
[1] 4
> y = 2*x # evaluates 2*x, i.e. 2*4, and stores the result in y.
> (y/x)^x
[1] 16
```

Side note: In many tutorials and R examples, you'll see `<-` (`alt+-` shortcut in Rstudio) instead of `=` as the assignment operator. It is perfectly acceptable (and more convenient, typing-wise), to use `=`. The `<-` operator came from a time when there was an actual button on many keyboards for that symbol. There are some very specific situations in which you'd need `<-` instead of `=`, but they won't come up at all in this class (and rarely, if ever, in your entire R career).

3.2 Character (text) and logical (TRUE or FALSE) variables.

Ok, so now we can assign values to variables, and do basic arithmetic with them. That's great for numbers, but there are other important types to let us deal with other kinds of data. Two that you'll use a lot are "character" and "logical" variables. A character can be anything in quotes, no matter how short or long.

```
name = "Winston"
date = "06-11-2015"
declarationOfIndependence = "When in the course of human events ..."
```

4 Logical variables

Logical (also sometimes called "boolean" variables), on the other hand, can only take one of two values, `TRUE` or `FALSE`, which are not quoted, but do have to be capitalized. We use logicals to represent something that

can take one of only two values. Importantly, they let us test for certain conditions between variables. For example, the expression `x>2` means "Is the value of x greater than 2?", and the result is either a `TRUE` or `FALSE`. Try it!

```
> x>2
[1] TRUE
```

4.1 Logical Operators

```
&   and # TRUE if both sides are TRUE
|   or  # TRUE if either side is TRUE
==  equal to (not to be confused with =, the assignment operator!)
!=  not equal to
>   greater than
>=  greater than or equal to
<   less than
<=  less than or equal to
```

Try these examples yourself!

```
x = 2 # Assign the value 2 to the variable x. Does the order matter? I.e. would 2=x be ok?
x == 3 # "Is the value of x equal to 3?" Does order matter here? Is 3==x ok?
day = "Monday"
day == "monday"
x>0 & (day=="Saturday" | day=="Sunday") # Is x positive and is it the weekend?
x<3 != 3>x # Why does this give an error?
(x<3) != (3>x) # Why does this fix it?
```

We can reverse the value of a logical with the `!` operator. `!TRUE` gives `FALSE` and vice versa.

```
> x = 2
> !(x==2) # x==2 is TRUE, and the negation of TRUE is FALSE
[1] FALSE
> !!(5!=4 | !(x<7)) # Why? Start with the innermost expressions and work your way outward.
[1] TRUE
```

To evaluate a complicated expression like the last one, don't work your way left to right. Rather, evaluate each component statement (like `5!=4`) in the console to make sure it's doing exactly what you think, and work your way incrementally from the inside out.

```
> 5!=4
[1] TRUE
> !(x<7)
[1] FALSE
> 5!=4 | !(x<7)
[1] TRUE
> !(5!=4 | !(x<7))
[1] FALSE
> !!(5!=4 | !(x<7))
[1] TRUE
```

5 If this, do that: the `if` and `if-else` statements

Very often we want to do one thing if a particular condition is true and some other thing otherwise, and logical operations make this possible. One construct for this that you'll use a lot is the if-statement, which looks like this:

```
if (condition) {
  code to executed if condition==TRUE
}

# For multiple conditions, there's if-else
if (condition1) {
  code to execute if condition1==TRUE
} else if (condition2) {
  code to execute if condition1==FALSE & condition2==TRUE
} else if (condition3) {
  code to execute if condition1==FALSE & conditon2==FALSE & condition3==TRUE
} .
  .
  .
} else {
  code to execute if all conditions are FALSE
}
```

The parentheses enclose the logical conditions being tested, while the braces `{}` enclose statements to be executed according if those conditions are met. A concrete example might help.

```
name = "Winston"
bday = "06-11"
date = "06-11"
if (date==bday) {
  print(paste("Happy Birthday,", name))
} else if (date=="12-25") {
  print(paste("Merry Christmas,", name))
} else {
  print(paste("Sorry,", name))
}
[1] "Happy Birthday, Winston"
```

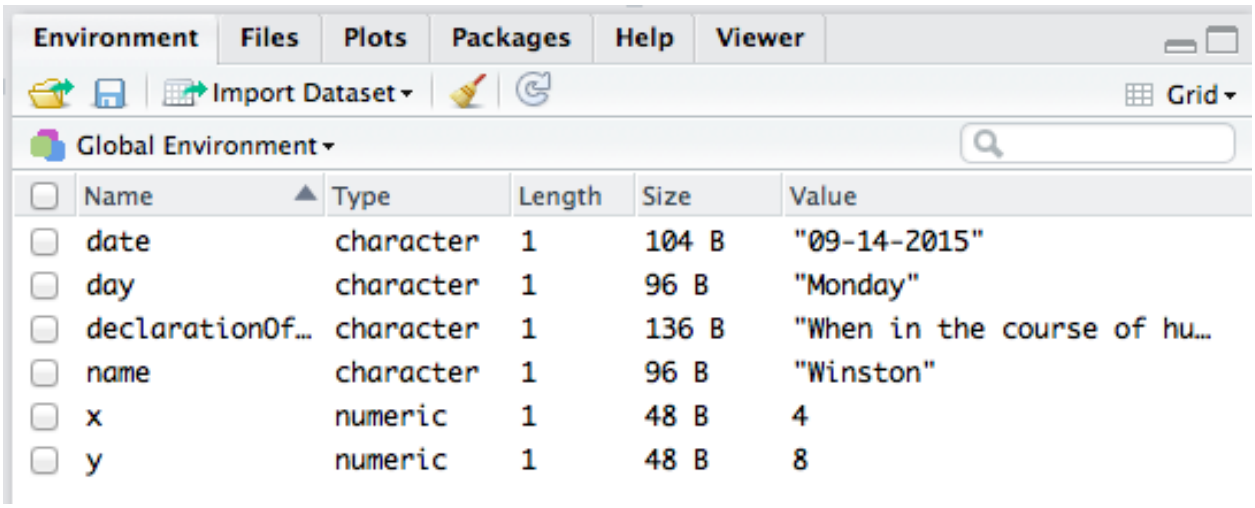
Change the value of `date` from `"06-11"` to `"12-25"`, and again to something else entirely, to make sure the code does what you expect.

5.1 Formatting: keep your code tidy!

Try typing the above code out in the editor, rather than just copying and pasting it. Note how Rstudio auto-completes parentheses and braces for you, and automatically indents when you hit Enter. Sticking to this indentation convention will a) help you to better read and understand your own code, and b) help your TA's to better read and understand your code, and a happy TA is a more partial-credit-awarding TA :).

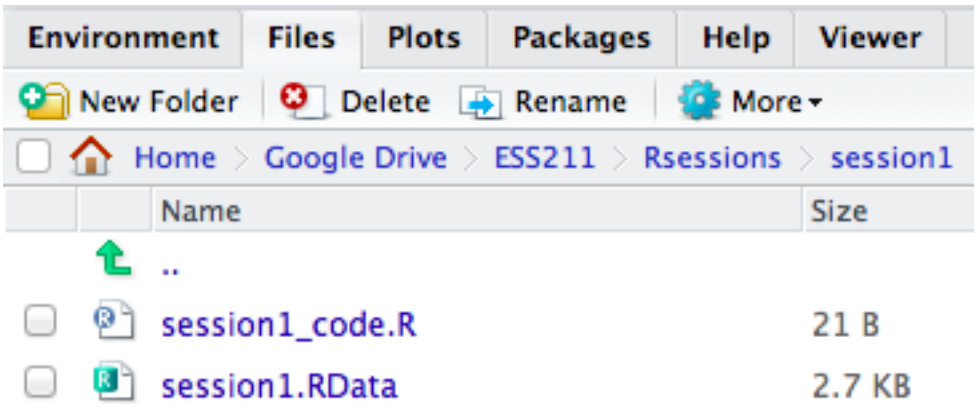
6 Saving and reloading your work

The "workspace" is the collection of all variables and their values in your current R session. Enter the command `ls()` to see a list of all variables currently in your workspace. Now, open the "Environment" tab in Rstudio, which lets you see the values associated with those variables, along with other helpful information.



6.1 Saving your workspace

When you end your R session (i.e. quit Rstudio), all those variables will be erased, and the next time you start Rstudio, a new R session begins with an empty workspace. This is one reason why you want to write code in the editor, so that you can save your code and re-generate your work. You can also save your actual workspace, not just the code that created it, by clicking the disk icon in the Environment tab. This will save an `.Rdata` file, in your working directory.



Sometimes there's a lot more junk in your workspace than you really care about. If you only want to save a few variables (say, `x` and `date`) to an `.Rdata` file, check only the boxes of those variables you want to save.

6.2 Loading an .Rdata file

After having saved an `Rdata` file, click the broom in the Environment tab, which should clear everything out of your workspace (in general, only do this if you're sure you've backed up your work!). Now, under the Files

tab, click the `.Rdata` file that you just created (and note the line of code that it executed in the console). Go back to Environment. Voila, they're all back!

6.3 Functions to do these things the "manual" way

Being able to browse files, set the working directory, see save and load files, etc. all from Rstudio's graphical interface is wonderful, but sometimes you'll need to do those same tasks within a script or from the console. Here's how:

```
# display the path of the current working directory
getwd()

# set the working directory
setwd("path/to/desired/working/directory")

# list all variables in the workspace
ls()

# list all files in the current working directory
list.files()

# list all files in some other directory
list.files("path/to/directory")

# save the variables var1, var2, etc. to an .Rdata file
save(var1,var2, ..., file="vars_to_save.Rdata")

# save all variables in the workspace
save.image(file="whole_workspace.Rdata")
# Alternatively, a dialog box should pop up when you quit Rstudio,
# asking whether you want to save your workspace.

# Load everything in this Rdata file into the workspace.
# Note: this assumes "my_rdata_file.Rdata" is in your working directory.
# Otherwise, you have to specify the full path.
load("my_rdata_file.Rdata")
```