Controlling and Managing Your R Workspace

This guide provides an overview of the R Workspace and R Environment—including its basic logic and structure. The goal is to develop an understanding of how best to control and manage your Workspace so that you can avoid or quickly diagnose several common errors.

Basic Overview

The R Environment consists of all the files necessary for running the R Program as well as data sets and other objects that you have created or loaded into your Workspace. These files can be broken down into three basic types:

1. The base packages that run all the standard analyses that we use in this course. These files are installed automatically when you first download and install the R program.
2. Additional packages you can install on your own and which allow for more advanced statistical analysis or additional commands.
3. The data sets that you download and other objects (data sets and variables) that you create.

We will not discuss the first two types in great detail. But, it is useful to take a brief look. Using the "search" command as follows, we can view our entire R Environment.

```
> search()

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

When you first install the R program, these are the 9 files you will find in your Environment. The first, ".GlobalEnv", is your "Global Environment" or R Workspace. It is where all your objects and data sets are stored. More on your Global Environment below. The remaining 8 files are required to run all the basic analyses—they will (or should) always be there.

You can install additional packages to do special types of analyses, not a part of the initial installation. For example, one common package is called "MASS". To install this package, simply type the following:

>library(MASS)

Typing in:

>search()

We can see how our R Environment has changed:

```
[1] ".GlobalEnv"        "package:MASS"      "package:stats"
[4] "package:graphics"  "package:grDevices" "package:utils"
```

```
[7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

Notice that everything is the same as above, except there is now a new file called "package:MASS". This is the file that you just installed with the "library(MASS)" command. Also, notice that it is in the 2nd position, just after the "Global Environment", which always stays in the first position.

Search Path:
The "search ()" command not only shows the entire R Environment; it also shows the order in which R accesses files—that is, it shows the *search path*. For example, ".GlobalEnv" (The Global Environment or R Workspace) is always first in the search path, while "package:base" is always last.

When you execute commands, R will always look first in the Global Environment for commands, data sets, and variables. If it does not find them there, it goes to the next file in the search path—in this case, "package:MASS". If it does not find them there, it will continue to the 3rd, 4th… all the way to the "package:base". If it does not find the object there, then guess what, you will get an error saying the "object is not found".

The Global Environment/R Workspace:
The global environment is the most important of all the files in the search path. This is where all your objects (data sets and variables) are stored—your R Workspace. To see your R Workspace, you need to use "ls()", which is short for "list".

>ls()

Since this lists all objects in your R Workspace, the output will vary from student to student. But, let's assume that you currently have 4 objects in your R Workspace: two variables called "name" and "age" and two data sets called "POL201" and "HDEV". Then the output will appear as follows:

[1] "age" "HDEV" "name" "POL201"

If a variable on its own (that is, not inside of a data set) is in our R Workspace, we can immediately run appropriate analyses on this variable. For example, we can calculate the mean for "age" simply as follows:

```
>mean(age)
[1] 22
```

ERROR ALERT
However, what happens when we try to run analyses on a variable inside of a data set? For example, there is a variable called "height" in the POL201 data set. If we try to take the mean of this variable (with the na.rm=T argument to remove missing values), we get the following:

```
>mean(height, na.rm=T)
Error in mean(height, na.rm = T) : object "height" not found
```

We get the error for the reason that is stated "object 'height' not found". The reason that the object is not found is that R can currently only see the data set, POL201, not its contents or variables.

Accessing Variables in Data Sets:

In order to access variables in our data sets, we have two options:

1. Use a $ sign to separate the name of the data set and the variable of interest. For example,

   ```
   >mean(POL201$height, na.rm=T)
   [1] 68.92958
   ```

   This tells R to first look at the data set, and then find the appropriate named variable inside of that data set.

   The one disadvantage of this approach is that involves more typing and longer commands. Thus, we can also opt to first:

2. Attach the data set with the "attach()" command:

   ```
   >attach(POL201)
   >mean(height, na.rm=T)
   [1] 68.92958
   ```

   * ERROR ALERT. Note: You can only attach data sets. You cannot attach the variables within the data set. For example, the following commands will simply give you error messages:

   ```
   > attach(height)
   Error in attach(height) : object "height" not found

   > attach(POL201$height)
   Error in attach(POL201$height) :
     'attach' only works for lists, data frames and environments
   ```

Attaching Data Sets and Changes to your R Environment:

When you attach a data set, R actually creates a copy of the data set and inserts it as a file in your R Environment. You can see this by re-examining your search path:

>search()

```
[1] ".GlobalEnv"        "POL201"            "package:MASS"
[4] "package:stats"     "package:graphics"  "package:grDevices"
[7] "package:utils"     "package:datasets"  "package:methods"
[10] "Autoloads"         "package:base"
```

You see that POL201 is now a file in your search path; more specifically, it is now the file in 2nd position in your search path and all other files have been shifted or "pushed down" one position.

You should now understand that attaching an object from your R Workspace, puts it in your R Environment, and then allows the contents of that data set (the individual variables) to be accessed for analysis.

You can remove a data set from your search path by detaching it:

```
>detach(POL201)
[1] ".GlobalEnv"        "package:MASS"      "package:stats"
[4] "package:graphics"  "package:grDevices" "package:utils"
[7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

Notice, POL201 is now gone from the search path.

Attaching Data Sets, Masking, and the Search Path:

It seems reasonable to ask at this point, why not simply attach all data sets and leave them attached? Often, this would work just fine. However, this will not work when:

1. You have an object in your Global Environment that has an identically name (and capitalization) to one in your data set of interest.
2. You have identically named variables in different data sets.

Both of these problems produce what are called "Masking" Errors. It is not technically an error in this sense that it won't allow you to run an analysis. R is simply telling you that one of your objects (variables) is being "masked" or covered up by another variable of the same name. This means you cannot actually access one of the same named variables for analysis.

Thus, the key question is which variable will it access for analysis? To answer this question, you must understand how R accesses data and files. You have already learned this: *it accesses files in the order of the search path.*

The most important thing here is that R always accesses objects in ".GlobalEnv" (the Global Environment) first. ***If you have a variable in your Global Environment that is identical to that in an attached data set, you will never be able to access and analyze that variable in the data set. It will always find the one in your Global Environment first.***

Let's create a masking error of this nature.

Starting with no data sets attached, let's create a new variable in your environment called "height".

```
>height = c(65, 70, 75)
```

Taking a look at our R Workspace, we now see 5 objects: the original 2 data sets and 2 variables as well as our newly created height variable.

```
> ls()
[1] "age"    "HDEV"   "height" "name"   "POL201"
```

Now, let's attach the POL201 data set and immediately you will see the masking problem.

```
> attach(POL201)

        The following object(s) are masked _by_ .GlobalEnv :

         height
```

Notice, it tells you directly that the object height is "masked_by_.GlobalEnv".

If we try to now take the mean of height, we will be taking the mean of the height variable from the Global Environment, not the one in the POL201 data set.

```
>mean(height, na.rm=T)
[1] 70
```

ERROR ALERT

Similarly, if we try to run a scatterplot of the relationship between "height" and "gpa", we will get an error message of a different sort.

```
> plot(height,gpa)
Error in xy.coords(x, y, xlabel, ylabel, log) :
  'x' and 'y' lengths differ
```

This is because again the height variable invoked here is not the one in the POL201 data set. The error occurs because the height variable in the global environment is not the same length as the "gpa" variable inside the POL201 data set.

Other Masking Problems:

Being "masked" by the Global Environment is the most common type of masking problem; however, there are a few others.

1. You attach the same data set twice. You can attach the same data set twice. It simply creates a new copy of the data set, places it at position #2, and pushes the old copy to position #3.

```
>attach(POL201)
The following object(s) are masked from POL201 ( position 3 ) :

         boycott career1 career2 colour country country1 country2 dist
do.study gender gpa haircut height idlgy internet job lang occupy party
petition protest shld.study smoke sport strike talk.pol transport tv
vote wkrhrs
```

Notice that it tells you that all the objects from POL201 in (position 3) are now masked, again meaning that they are unavailable for analysis.

It should be clear that this is not really a big deal, because those variables are being masked by an exact copy of the other variables.

You can see the basic logic of double attaching by examining your search path.

```
>search()
[1] ".GlobalEnv"        "POL201"            "POL201"
[4] "package:MASS"      "package:stats"     "package:graphics"
[7] "package:grDevices" "package:utils"     "package:datasets"
[10] "package:methods"   "Autoloads"         "package:base"
```

You see "POL201" is now in two unique positions: Position #2 and Position #3.

2. The second type of masking problem occurs when you attach two different data sets which happen to have an identically named variable. This might happen once or twice throughout the semester. The solution is to simply detach the data set that is not of interest.

Detaching Data Sets:
To detach a data set you simply use the "detach()" command. For example,

```
>detach(POL201)
```