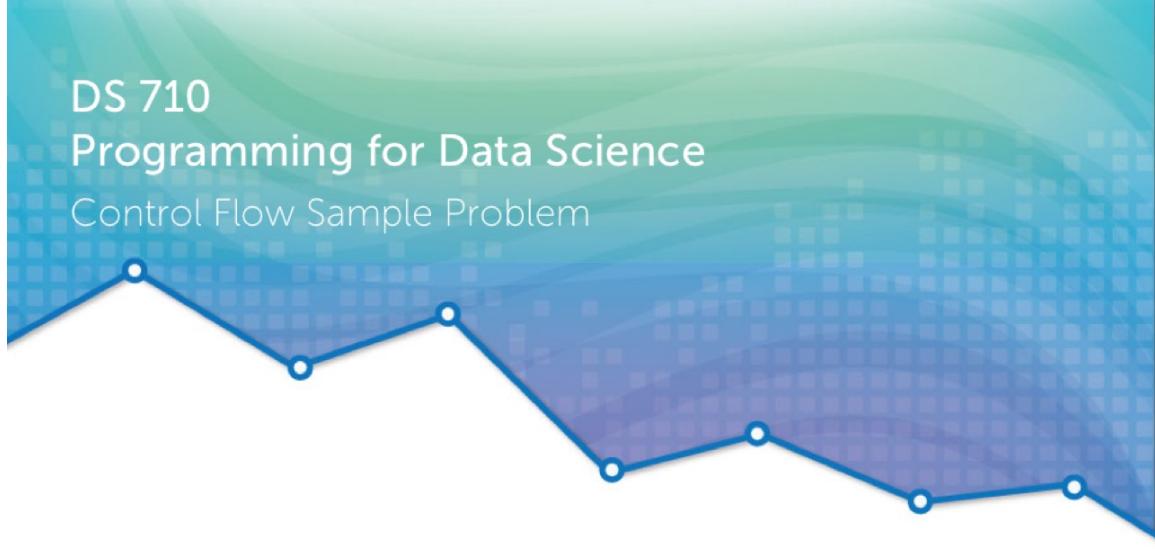


DS 710

Programming for Data Science

Control Flow Sample Problem



Analyzing a Data Set in R

Control Flow sample problem:
Analyzing a data set in R

Part 1

Let's look at how to analyze a sample problem involving analyzing a data set using control flow in R.

Sample Problem

- a. Download **Cars 2005.csv**, load the data into R, and attach it.
- b. Print the types (hatchback, sedan, etc.) of all the cars with prices less than \$10,000.
- c. Count the number of cars with prices less than \$10,000.

We'll analyze the data set `cars2005.csv`. First we'll need to load the data into R and attach it. Then we'll print the types of all the cars with prices less than \$10,000. Finally we'll count how many cars have prices less than \$10,000.

.csv files

- Comma separated values
- Can be opened in many different ways
 - Basic text editor
 - Excel
 - R

CSV stands for comma separated values. This is a nice file format, because it can be opened in many different ways. Using a basic text editor, Excel, or R.

Basic text editor

```
Price,Mileage,Make,Model,Trim,Type,Cylinder,Liter,Doors,Cruise,Sound,Leather  
17314.10313,8221,Buick,Century,Sedan 4D,Sedan,6,3.1,4,1,1,1  
17542.03608,9135,Buick,Century,Sedan 4D,Sedan,6,3.1,4,1,1,0  
16218.84786,13196,Buick,Century,Sedan 4D,Sedan,6,3.1,4,1,1,0  
16336.91314,16342,Buick,Century,Sedan 4D,Sedan,6,3.1,4,1,0,0  
16339.17032,19832,Buick,Century,Sedan 4D,Sedan,6,3.1,4,1,0,1
```

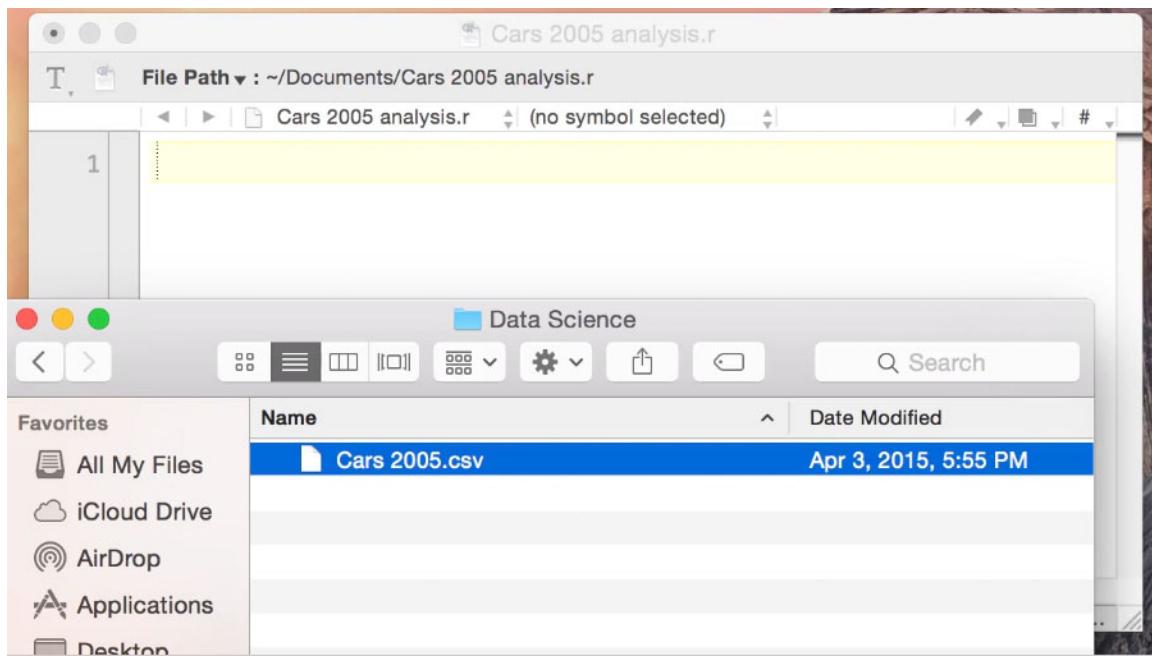
If we open the car's2005 data set in a basic text editor, such as notepad++ we can see that each of the entries in the data set is separated by commas. Hence, comma separated values.

Excel

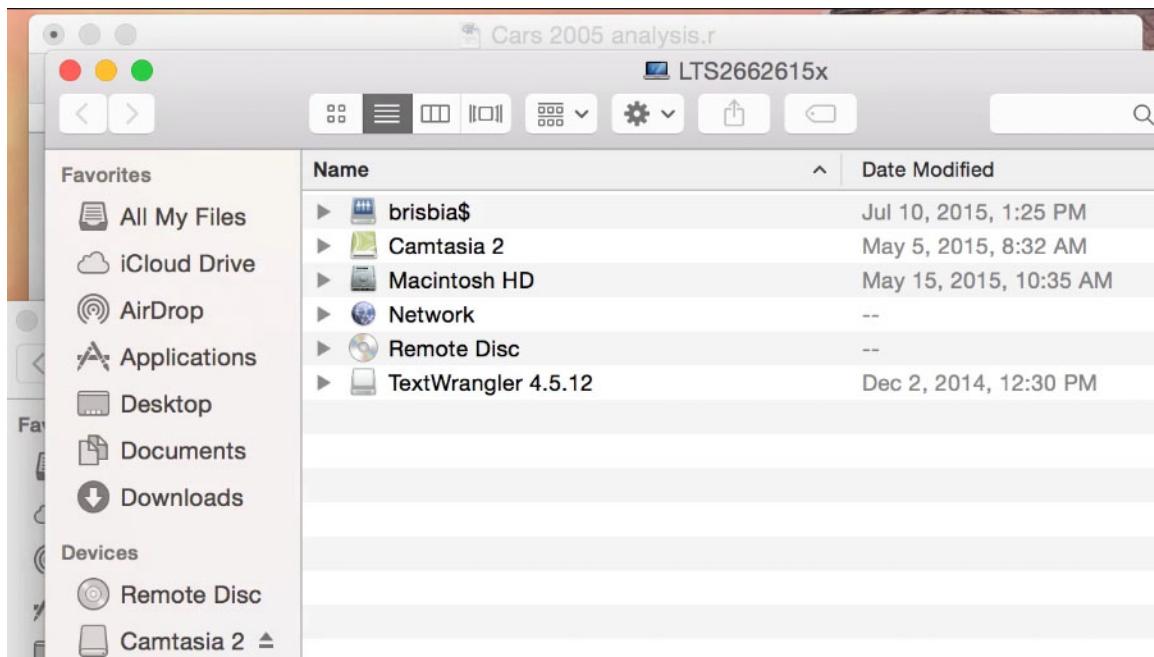
A	B	C	D	E	F	G	H	I	J	K	L	
1	Price	Mileage	Make	Model	Trim	Type	Cylinder	Liter	Doors	Cruise	Sound	Leather
2	17314.1	8221	Buick	Century	Sedan 4D	Sedan	6	3.1	4	1	1	1
3	17542.04	9135	Buick	Century	Sedan 4D	Sedan	6	3.1	4	1	1	0
4	16218.85	13196	Buick	Century	Sedan 4D	Sedan	6	3.1	4	1	1	0
5	16336.91	16342	Buick	Century	Sedan 4D	Sedan	6	3.1	4	1	0	0
6	16339.17	19832	Buick	Century	Sedan 4D	Sedan	6	3.1	4	1	0	1

- Useful for an initial view of the data
- Too slow for very large data sets

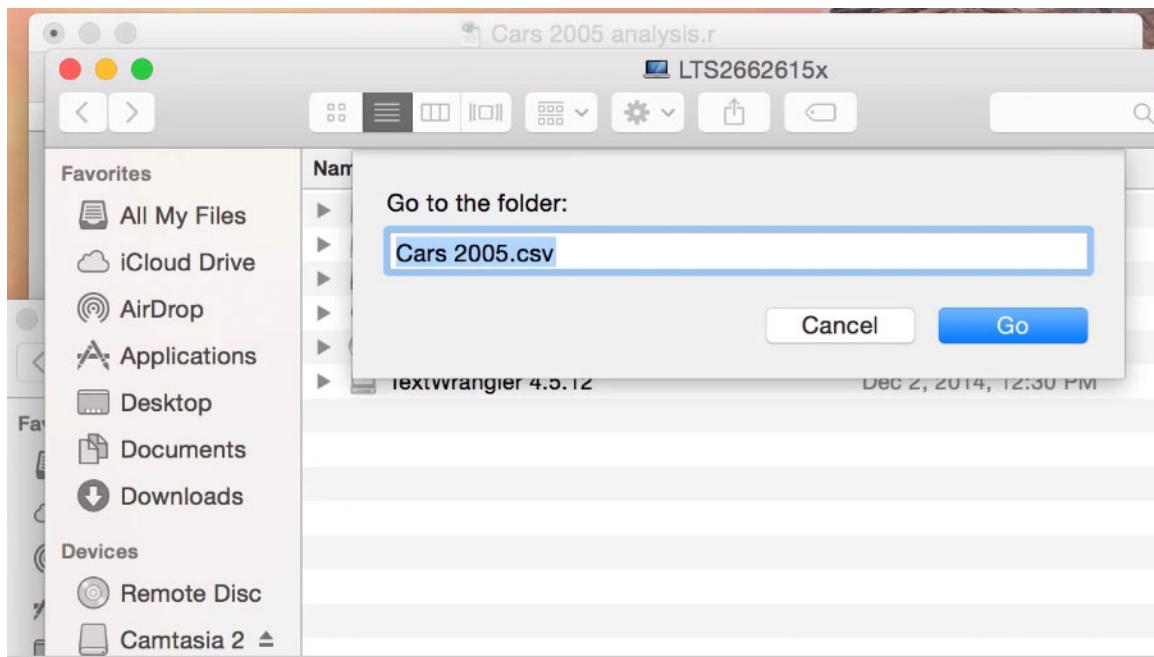
If we open it in Excel, we can see that each variable is separated into its own column. This is useful for getting an initial view of what's going on with the data set, at least if your data set isn't too large. In this case, we can see that the data contains information on used cars. If you're curious about the details of what each column contains, you can check out the metadata using the link below. We can also see that this data set contains an initial row that tells what each column contains, this is called a header row. Next, let's take a look at how to read this data set into R.



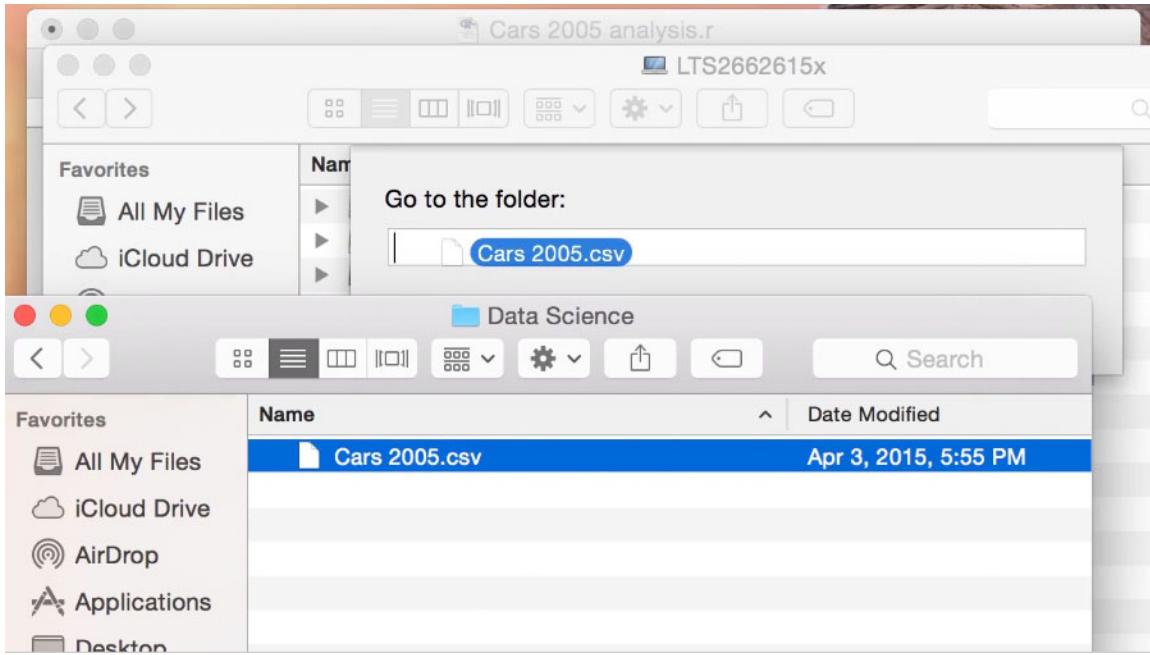
To open a CSV file in R, we need to make sure we know the full path of the directory where the file is saved. To find this on a Mac open the finder, and navigate to the file.



Then press Command N to open another finder window.

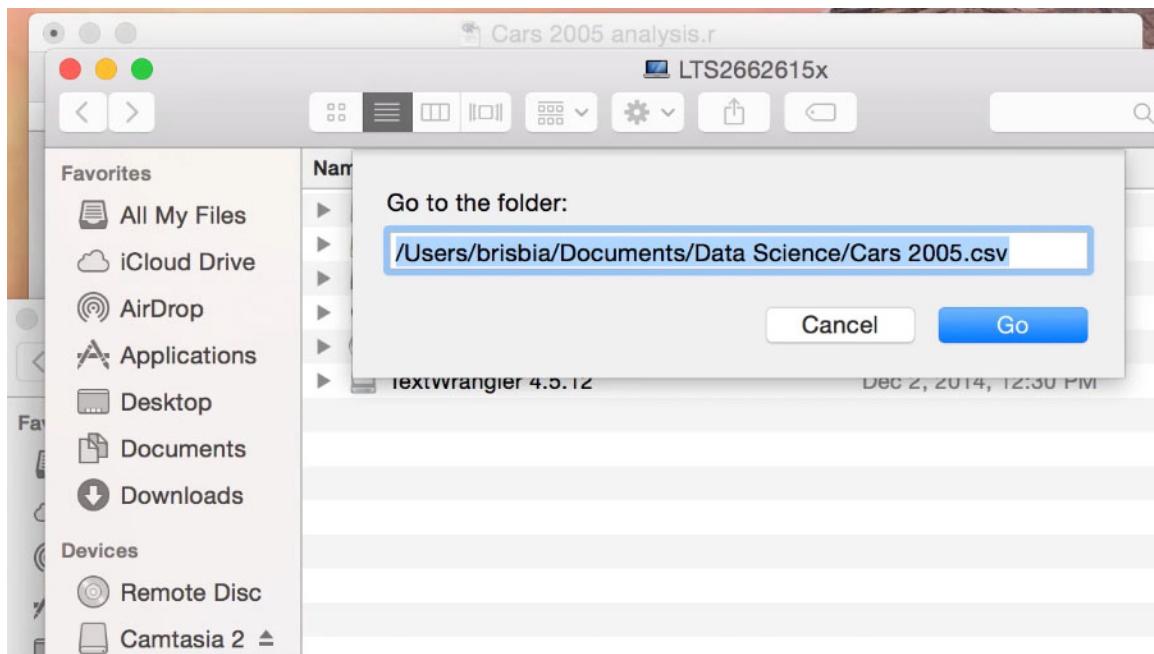


Then press Shift Command G to open the go to Window. Delete anything that's written on this line.

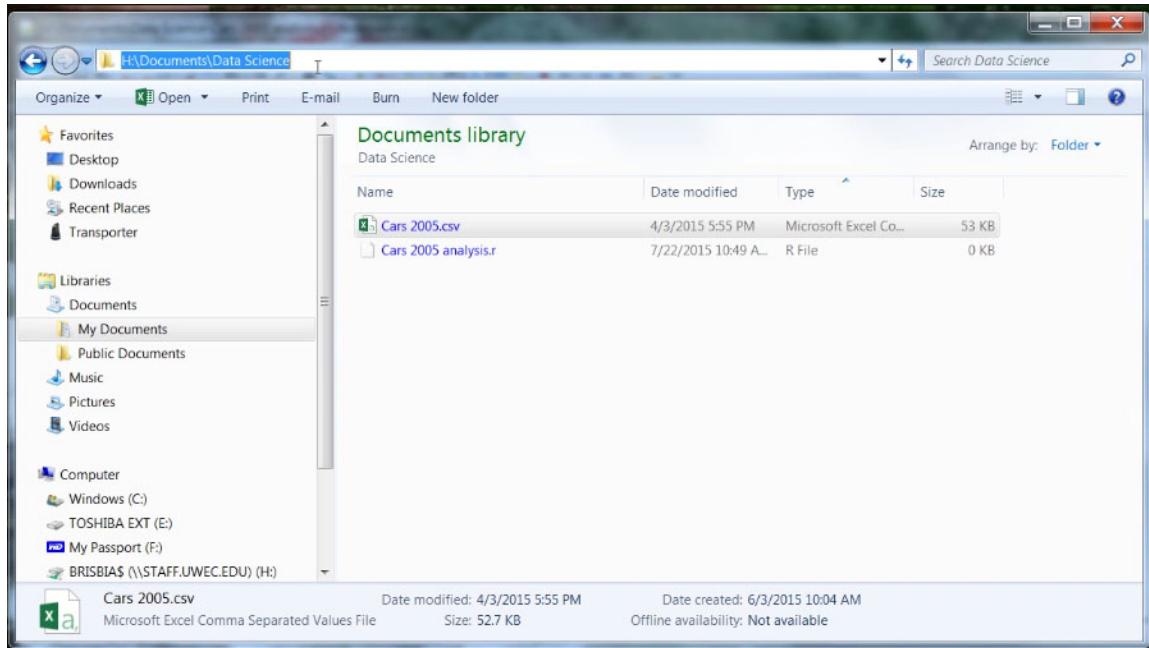


Then go back to our original finder window and click and drag that file into the go to Window. This shows you the full path of where the file is located.

In my case you can see that I've saved the file in the data science folder, or directory, which in turn is in the Documents folder. The full path of a file on a Mac should always begin with a forward slash.

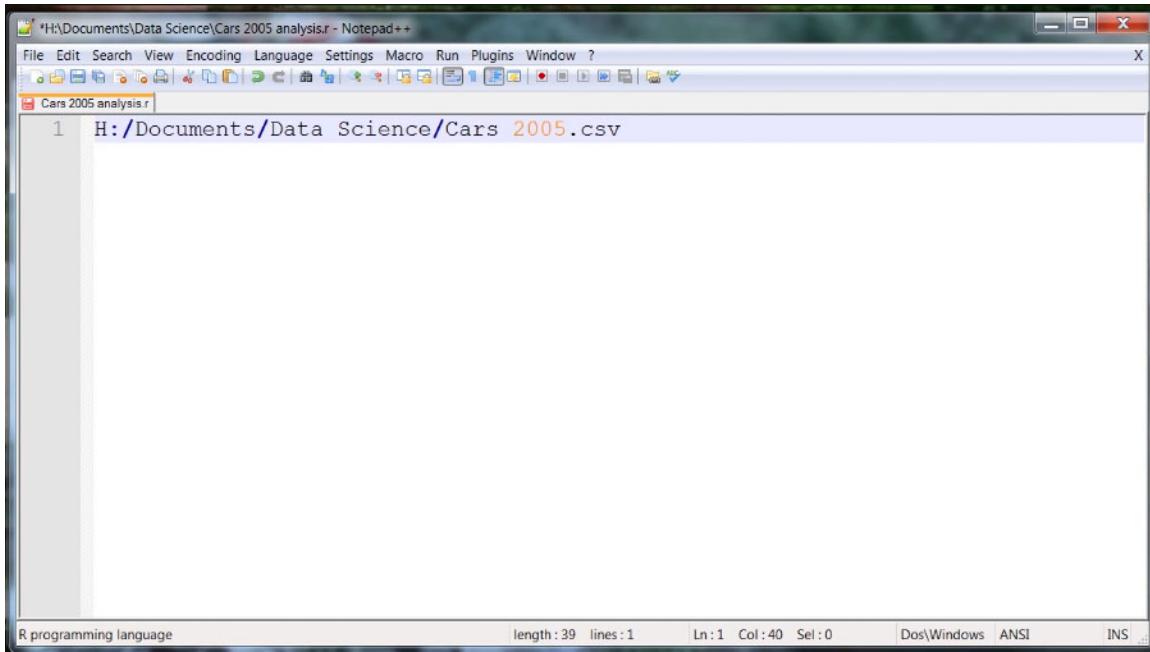


Highlight this path, and then press Command C to copy it, and switch over to your R script. Here I've saved my R script as a .R file in Text Wrangler. So everything I type will be color-coded. Press Command V to copy and paste our file name.



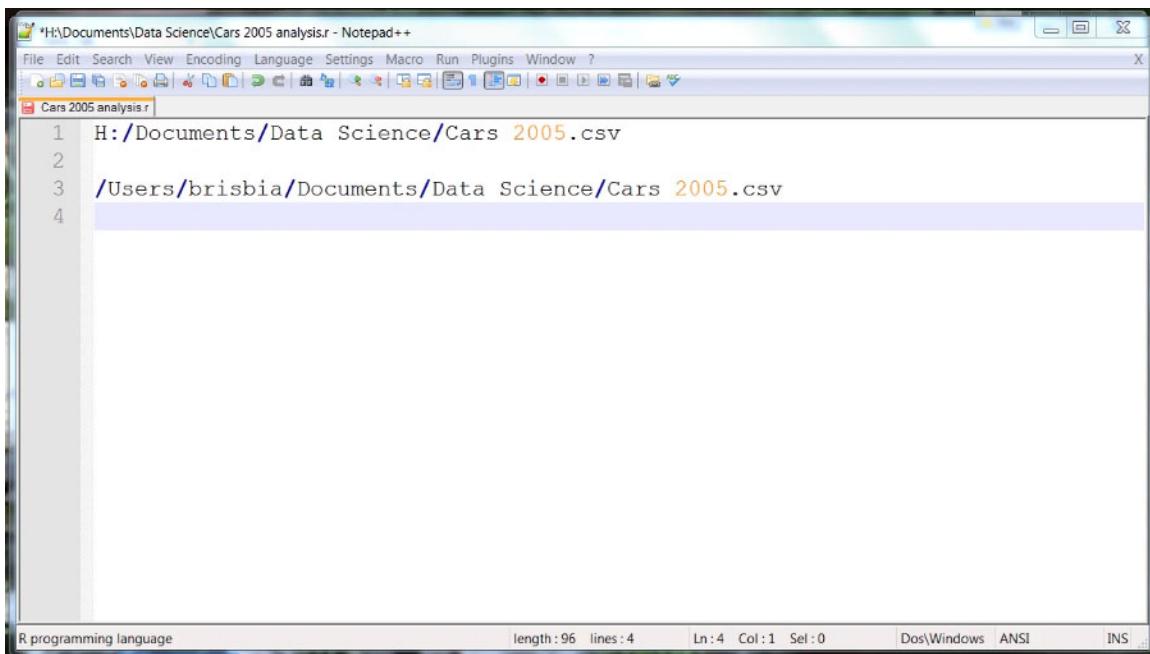
To open a CSV file in R we need to make sure we know the full path of the directory where the file is saved. To find this in windows open the file manager and navigate to where you can see the file. Then click on the address bar in the upper part of the window. Here you can see the full path of where the file is saved. In my case, I've saved it in the data science directory, or folder, which is in the documents directory, which is on the H drive.

In Windows, file paths should always start with a letter and a colon, that generally means that you don't want to save files directly to the desktop. Highlight the full file path and press Control C to copy it.



Then switch over to your R script. I've saved my R script as a .R file in Notepad++. That means that everything I type will be color-coded.

Next I'll press Control V to paste the file path. There's one more thing that we need to fix, and that is that Windows uses backslashes in its file paths, but R interprets backslashes as a signal to ignore whatever letter comes next. We don't want that, so we need to change the backslashes to forward slashes. Finally, add another forward slash and type the name of the file.



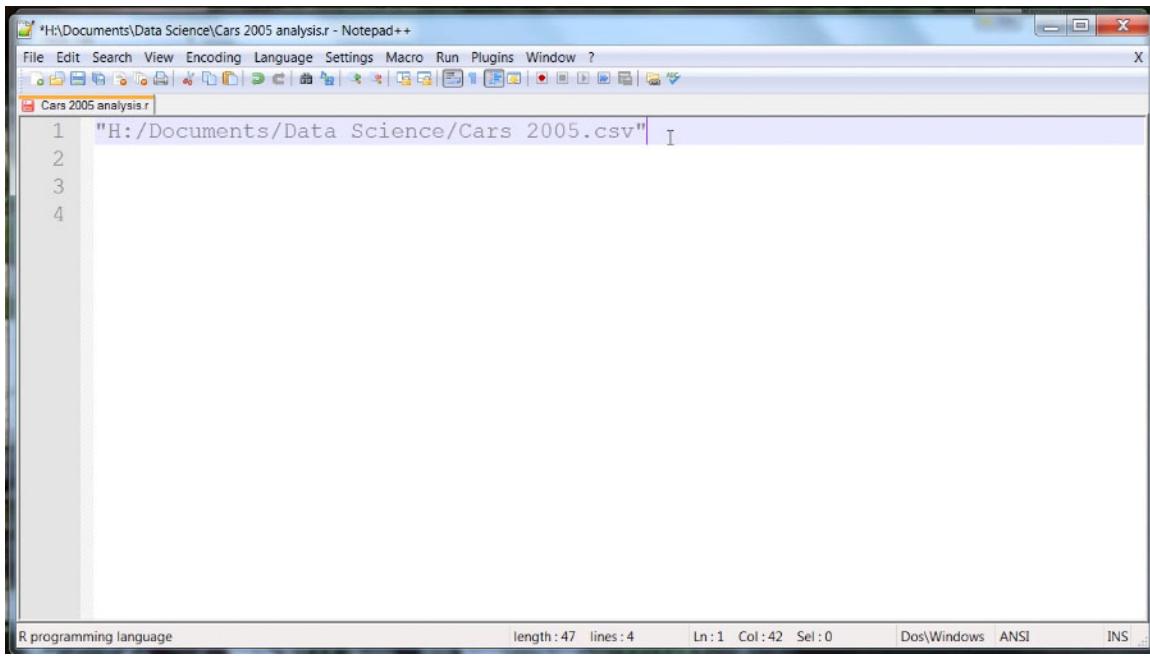
The screenshot shows a Notepad++ window titled "H:\Documents\Data Science\Cars 2005 analysis.r - Notepad++". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, Window, and ?.

The code in the editor is:

```
1 H:/Documents/Data Science/Cars 2005.csv
2
3 /Users/brisbia/Documents/Data Science/Cars 2005.csv
4
```

The status bar at the bottom indicates "R programming language", "length : 96", "lines : 4", "Ln : 4 Col : 1 Sel : 0", "Dos\Windows ANSI", and "INS".

Now we have the full path of the file we want to read into R, written in an R script in either Notepad++ in Windows or Text Wrangler in Mac. This will look a little different depending on whether you're working on a Windows machine or on a Mac, but the rest of what we're going to do will be the same regardless of what kind of operating system you're using.



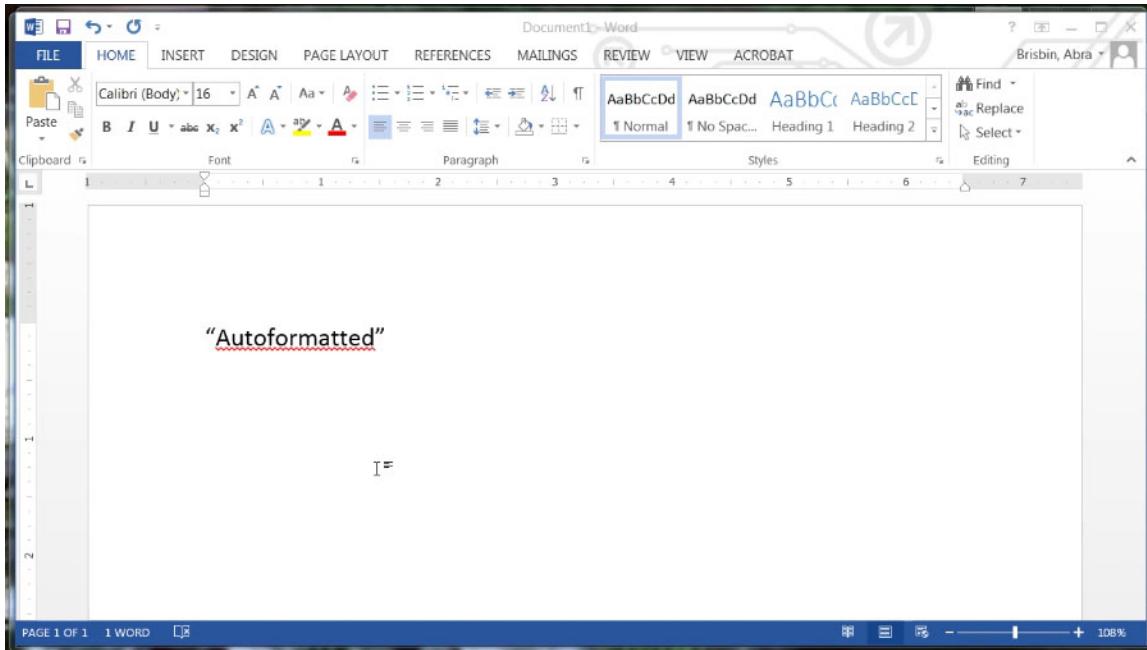
The screenshot shows a Notepad++ window titled "H:\Documents\Data Science\Cars 2005 analysis.r - Notepad++". The code in the editor is:

```
1 "H:/Documents/Data Science/Cars 2005.csv" I  
2  
3  
4
```

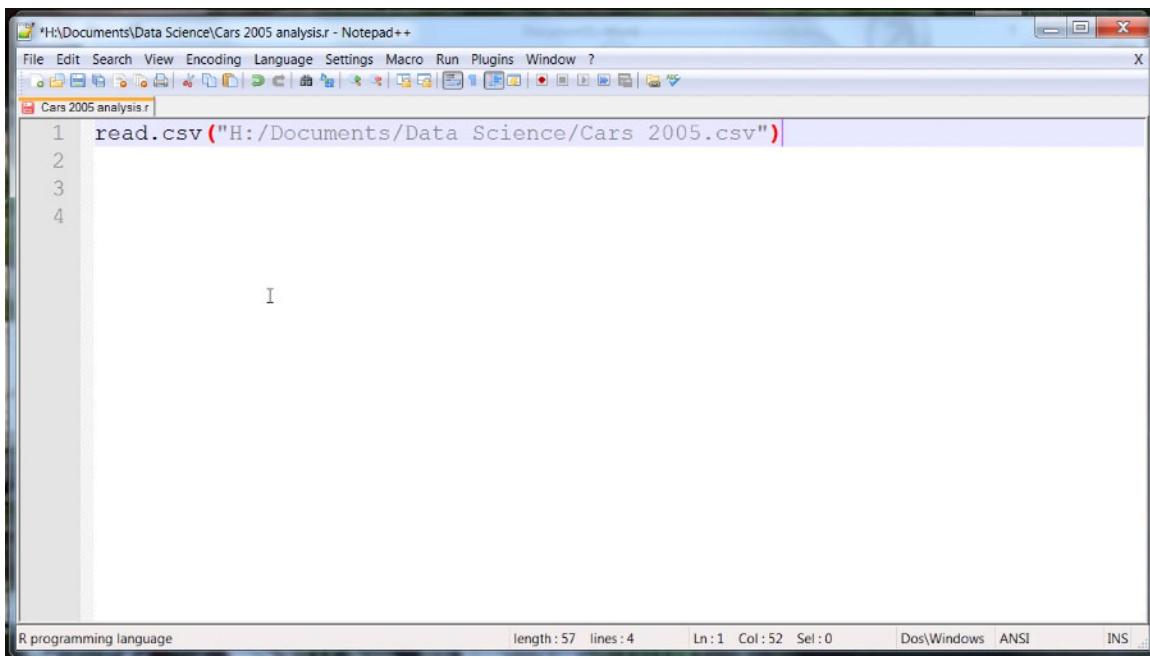
The status bar at the bottom indicates "R programming language", "length : 47", "lines : 4", "Ln : 1", "Col : 42", "Sel : 0", "Dos\Windows", "ANSI", and "INS".

I'm using a Windows computer, so I'll delete the file path on Mac, and work with the file path in Windows, and I've already made sure that all of my slashes are forward slashes not back slashes.

You want to start by enclosing the file path in quotation marks. This tells R that the file path is a string, or basically a series of words. It's not the name of a variable. You want to make sure that your quotation marks are straight vertical quotation marks, not auto formatted to indicate a difference between the quotation marks at the start, versus the end of the quote.



For example, the quotation marks in a Word document are auto formatted, and R doesn't understand how to interpret those. This won't be an issue if you're using Notepad++ or Text Wrangler, but it can be a problem if you're trying to copy code from a Word document. On a Mac if you're using text edit instead of Text Wrangler hold down the Control key when you're typing the quote mark, that will prevent auto formatting.

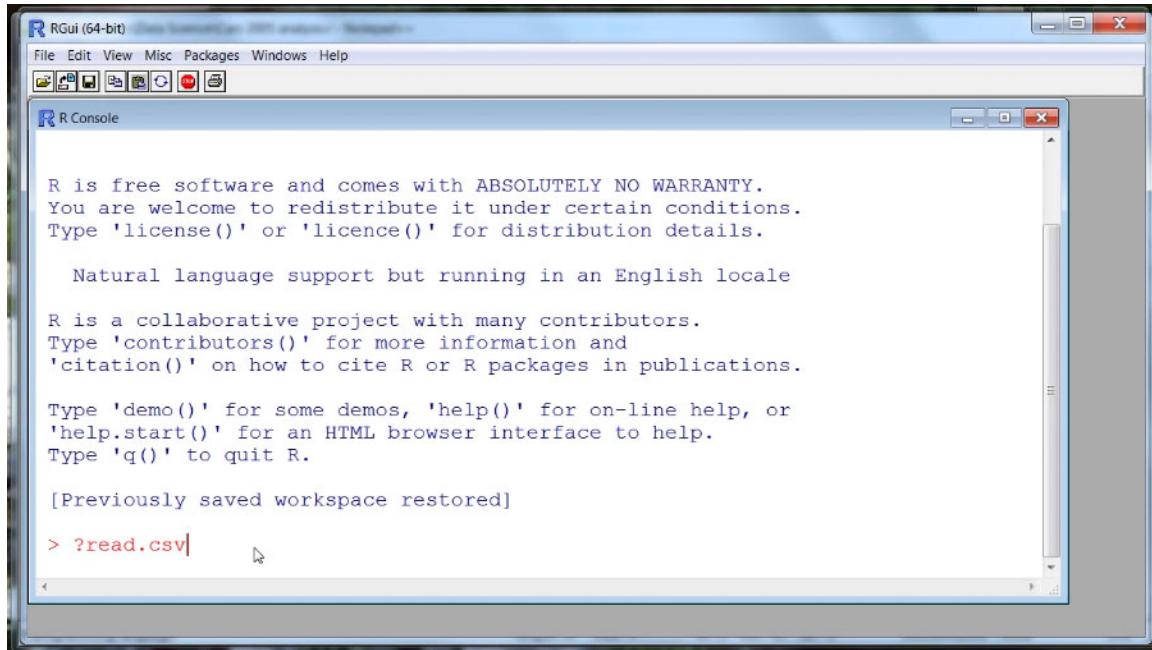


A screenshot of the Notepad++ text editor. The title bar reads "H:\Documents\Data Science\Cars 2005 analysis.r - Notepad++". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, Window, and ?. The toolbar below the menu has various icons for file operations like Open, Save, Find, and Print. The main code editor window contains the following R code:

```
1 read.csv("H:/Documents/Data Science/Cars 2005.csv")
```

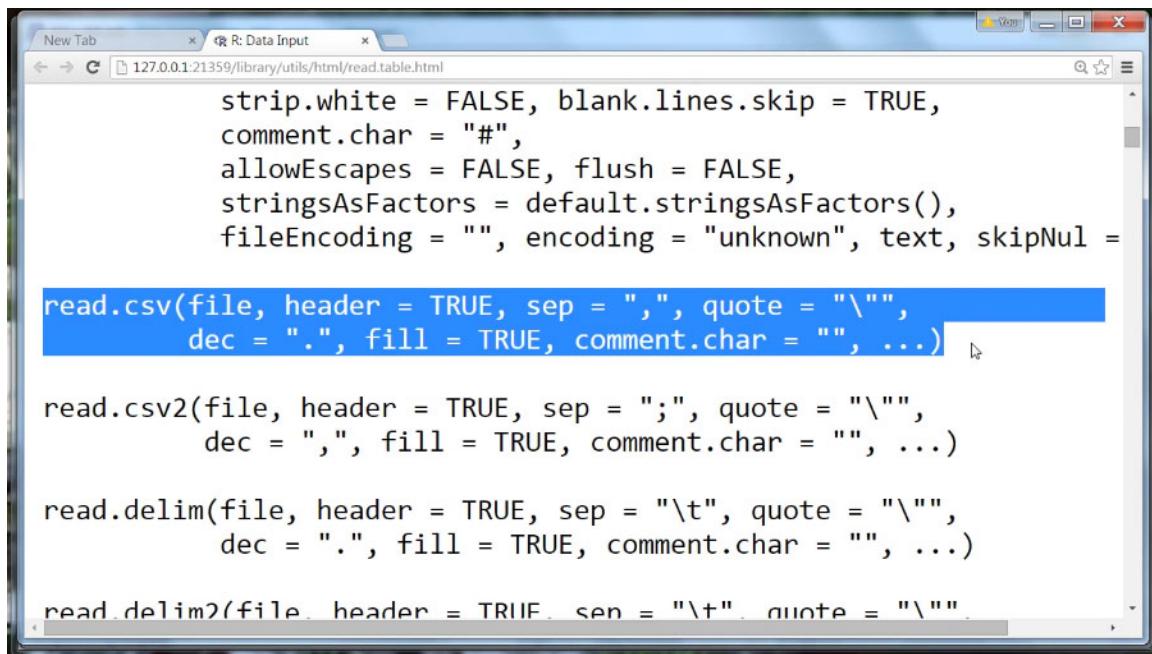
The status bar at the bottom shows "R programming language", "length : 57 lines : 4", "Ln : 1 Col : 52 Sel : 0", "Dos\Windows ANSI", and "INS".

All right, to read the CSV file into R we want to use the `read.csv` function. That's `read.csv`, not `cvs`. Remember CSV stands for comma separated values, and for most functions in R we enclose the arguments in parentheses.



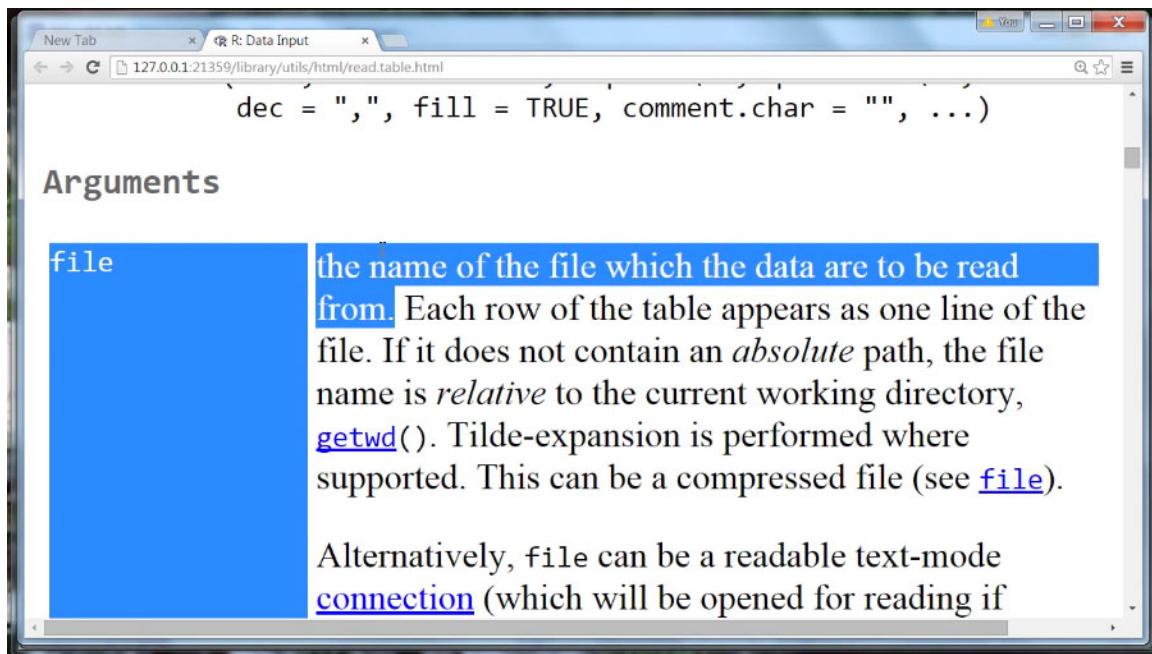
If you don't remember what a particular function does, you can always look it up by going to the R console, and typing a question mark and then the name of the function.

We press Enter, and this brings us to the manual page for this function. I'll make it a little bigger so it's easier to read.

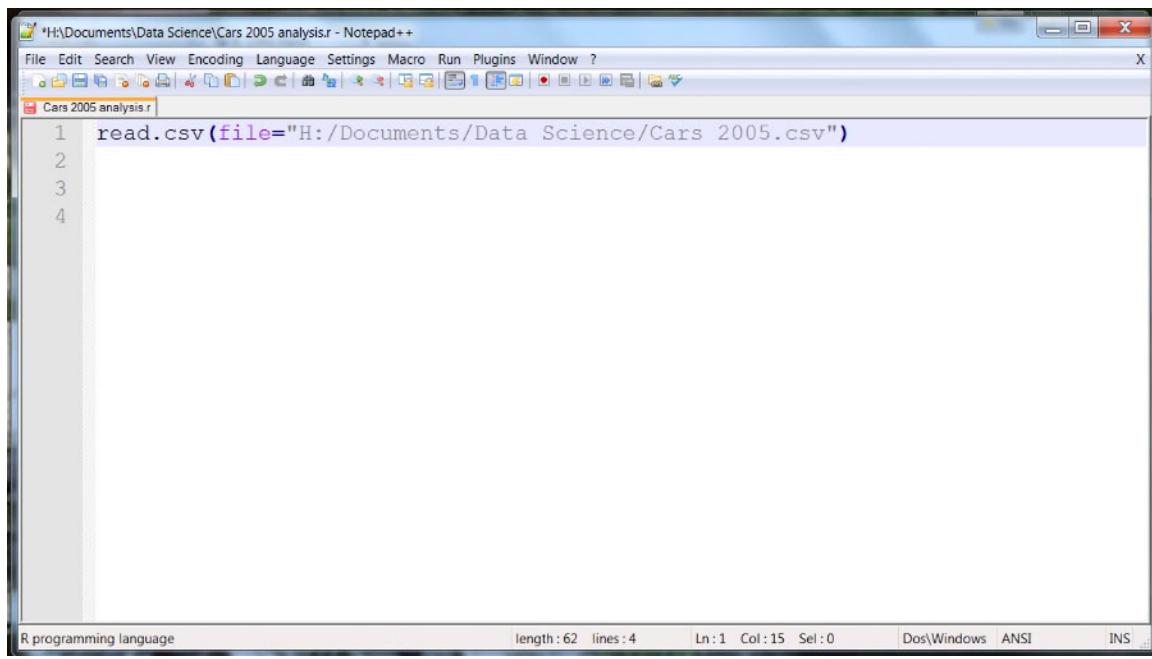


```
strip.white = FALSE, blank.lines.skip = TRUE,
comment.char = "#",
allowEscapes = FALSE, flush = FALSE,
stringsAsFactors = default.stringsAsFactors(),
fileEncoding = "", encoding = "unknown", text, skipNul =
read.csv(file, header = TRUE, sep = ",", quote = "\"",
          dec = ".", fill = TRUE, comment.char = "", ...)
read.csv2(file, header = TRUE, sep = ";", quote = "\"",
          dec = ",", fill = TRUE, comment.char = "", ...)
read.delim(file, header = TRUE, sep = "\t", quote = "\"",
           dec = ".", fill = TRUE, comment.char = "", ...)
read.delim2(file, header = TRUE, sep = "\t", quote = "\"",
            dec = ",", fill = TRUE, comment.char = "", ...)
```

Here you can see the syntax that you use to use the function read.csv, and the first argument is file.



If we scroll down, we can read the description of that argument, and see that it's the name of the file from which the data are to be read from. That's what we want. So our file path is listed correctly as the first argument of `read.csv`.

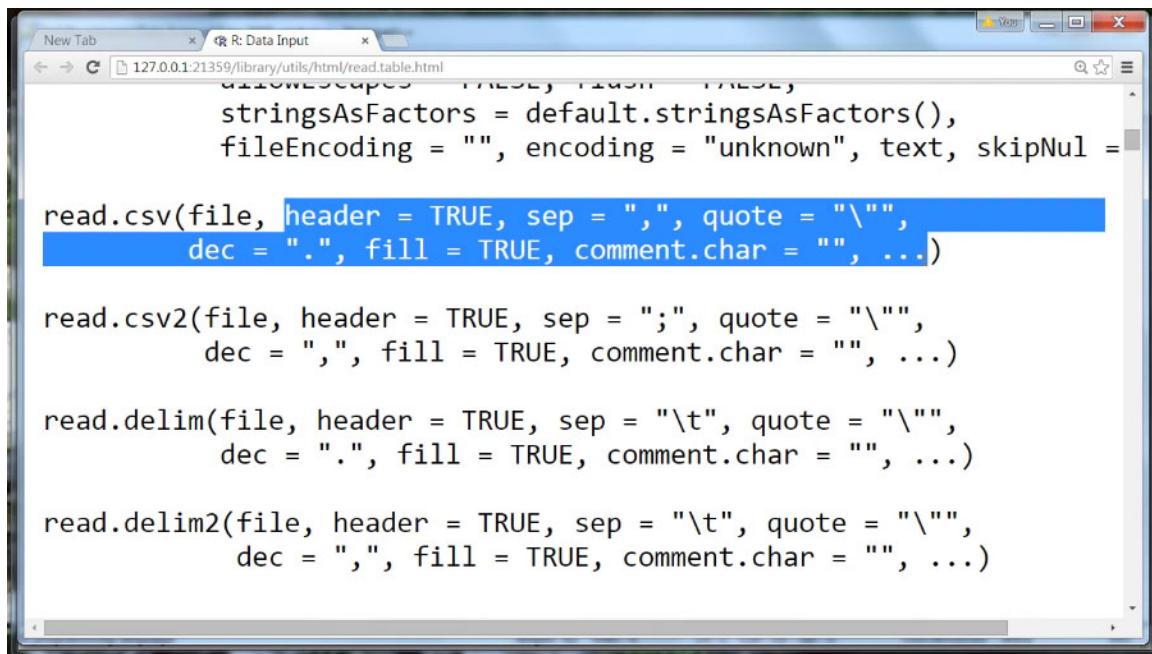


A screenshot of the Notepad++ text editor. The title bar reads "H:\Documents\Data Science\Cars 2005 analysis.r - Notepad++". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, Window, and ?. The toolbar below the menu has various icons for file operations like Open, Save, Print, and Find. The main code editor window contains the following R code:

```
1 read.csv(file="H:/Documents/Data Science/Cars 2005.csv")
```

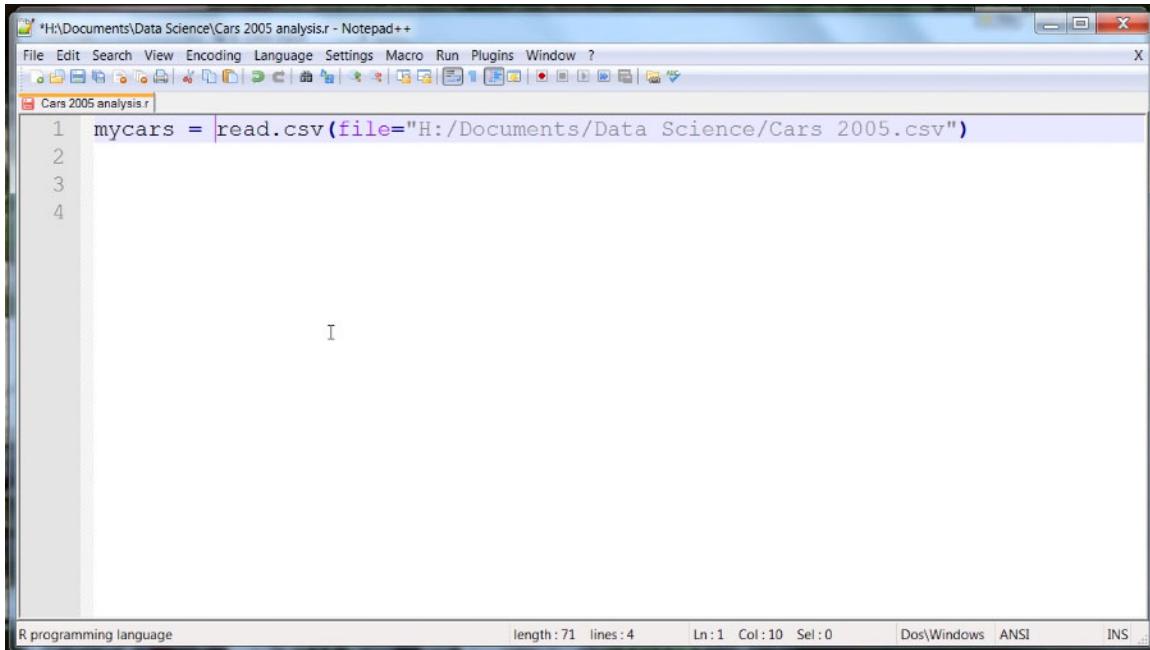
The status bar at the bottom shows "R programming language", "length : 62 lines : 4", "Ln : 1 Col : 15 Sel : 0", "Dos\Windows ANSI", and "INS".

If we want we can be specific by typing file equals, and then the file name. But this is optional.



```
allowEscapes = FALSE, na.strings = NA_real_,  
stringsAsFactors = default.stringsAsFactors(),  
fileEncoding = "", encoding = "unknown", text, skipNul =  
  
read.csv(file, header = TRUE, sep = ",", quote = "\"",  
         dec = ".", fill = TRUE, comment.char = "", ...)  
  
read.csv2(file, header = TRUE, sep = ";", quote = "\"",  
          dec = ",", fill = TRUE, comment.char = "", ...)  
  
read.delim(file, header = TRUE, sep = "\t", quote = "\"",  
           dec = ".", fill = TRUE, comment.char = "", ...)  
  
read.delim2(file, header = TRUE, sep = "\t", quote = "\"",  
            dec = ",", fill = TRUE, comment.char = "", ...)
```

You'll notice that `read.csv` takes several other possible arguments, but all of them have equals something after the argument. Those are the default values that R will use if you don't type that argument in the function call. For our purposes, the default values are fine, so we don't have to type anything else.



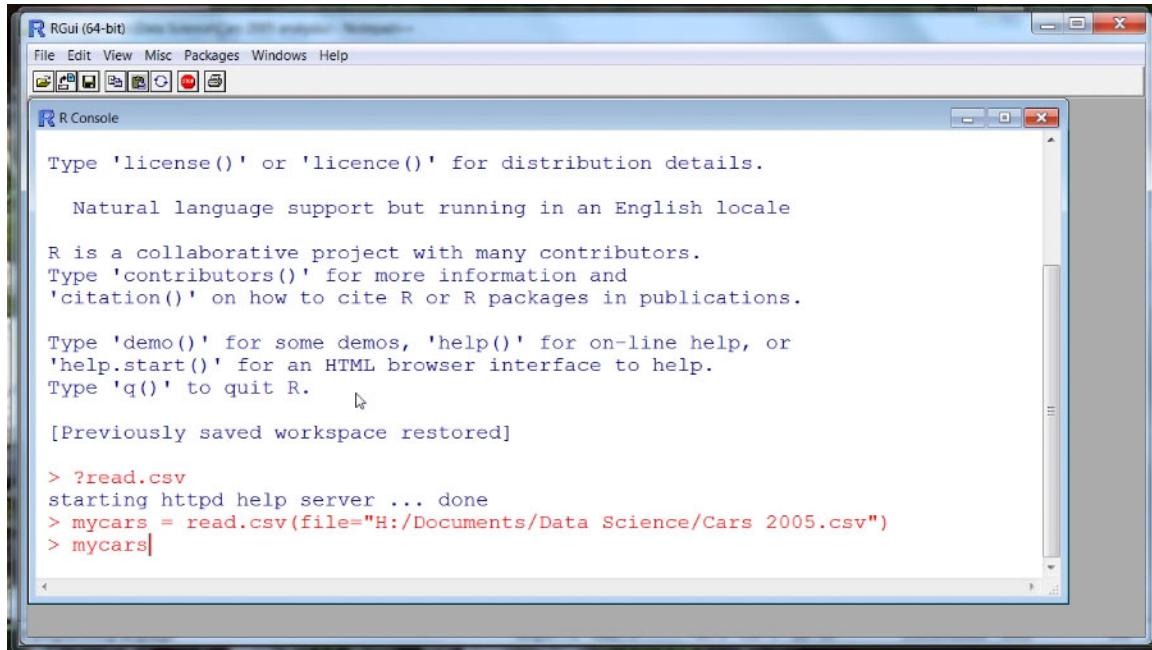
The screenshot shows a Notepad++ window titled "H:\Documents\Data Science\Cars 2005 analysis.r - Notepad++". The code in the editor is:

```
1 mycars = read.csv(file="H:/Documents/Data Science/Cars 2005.csv")
```

The status bar at the bottom indicates: R programming language, length : 71, lines : 4, Ln : 1, Col : 10, Sel : 0, Dos\Windows, ANSI, INS.

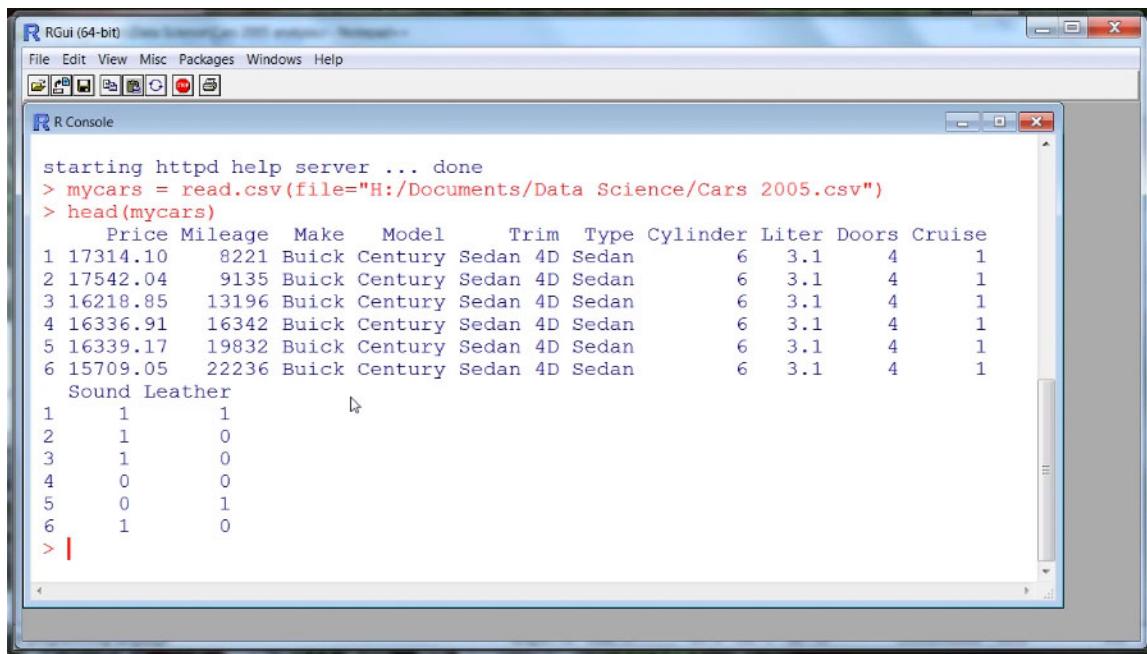
This is now a perfectly valid line of R code. But if we tried entering this into the R console, R would read in the data and print it right back out to the screen without storing it anywhere in memory.

We'd like to store R data in memory, so we need to give this a variable name. Say mycars equals read.csv. You can get any valid variable name you like, but to avoid confusion later, it's good to choose a variable name that's not the same as any of the column headers in the data set.



To run this code, we can copy it by pressing Control C, or Command C, and then paste it into the R console using Control V or Command V, and press Enter.

You'll notice that nothing happens on the screen. In order to see the data that we've just read in, we need to type the name of the variable where we sort it, and press Enter. But this is a fairly large data set, so it's more convenient just to look at the first few lines of code.



The screenshot shows the RGui (64-bit) interface with an R Console window open. The console displays the following R code and its output:

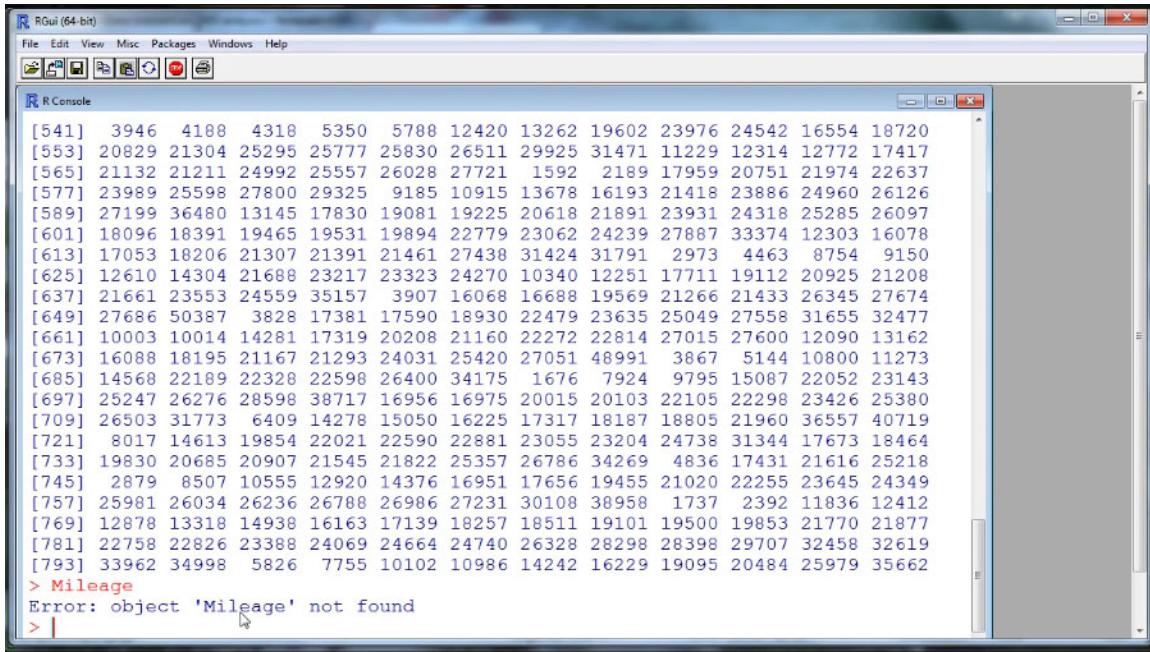
```
starting httpd help server ... done
> mycars = read.csv(file="H:/Documents/Data Science/Cars 2005.csv")
> head(mycars)
   Price Mileage Make Model Trim Type Cylinder Liter Doors Cruise
1 17314.10    8221 Buick Century Sedan 4D Sedan      6   3.1     4     1
2 17542.04    9135 Buick Century Sedan 4D Sedan      6   3.1     4     1
3 16218.85   13196 Buick Century Sedan 4D Sedan      6   3.1     4     1
4 16336.91   16342 Buick Century Sedan 4D Sedan      6   3.1     4     1
5 16339.17   19832 Buick Century Sedan 4D Sedan      6   3.1     4     1
6 15709.05   22236 Buick Century Sedan 4D Sedan      6   3.1     4     1
  Sound Leather
1       1         1
2       1         0
3       1         0
4       0         0
5       0         1
6       1         0
> |
```

We can do that using the `head` function, and press Enter. Here you can see the first six rows of data in all of the columns. In this case my window wasn't quite wide enough so the last two columns wrapped around and they're listed below the first two columns.

```
RGui (64-bit)
File Edit View Misc Packages Windows Help
R Console
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> ?read.csv
starting httpd help server ... done
> mycars <- read.csv(file="H:/Documents/Data Science/Cars 2005.csv")
> head(mycars)
   Price Mileage Make Model Trim Type Cylinder Liter Doors Cruise
1 17314.10    8221 Buick Century Sedan 4D Sedan      6   3.1     4     1
2 17542.04    9135* Buick Century Sedan 4D Sedan      6   3.1     4     1
3 16218.85   13196 Buick Century Sedan 4D Sedan      6   3.1     4     1
4 16336.91   16342 Buick Century Sedan 4D Sedan      6   3.1     4     1
5 16339.17   19832 Buick Century Sedan 4D Sedan      6   3.1     4     1
6 15709.05   22236 Buick Century Sedan 4D Sedan      6   3.1     4     1
   Sound Leather
1        1        1
2        1        0
3        1        0
4        0        0
5        0        1
6        1        0
> mycars[1,2]
[1] 8221
> |
```

Our data set is now stored as a matrix. We can refer to particular elements of the matrix using square bracket notation. For example, mycars, square bracket, 1 comma 2 will give the value in the first row and second column. For example, 8,221 we can see was in the first row and second column when we looked at the first few lines of data. This means that we could look at the whole second column by typing mycars, square bracket, comma 2. When we don't put anything in the rows section of the square bracket notation, then we look at all the rows.

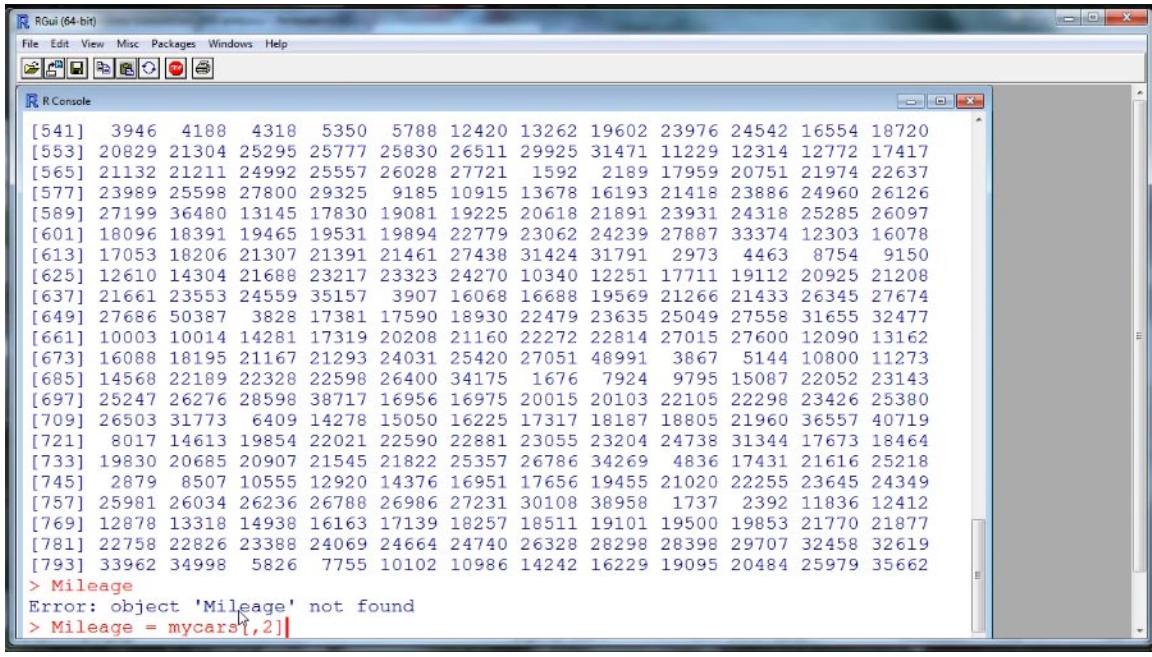


The screenshot shows the RGui (64-bit) interface. The title bar reads "RGui (64-bit)". The menu bar includes "File", "Edit", "View", "Misc", "Packages", "Windows", and "Help". Below the menu is a toolbar with various icons. The main window is titled "R Console". It displays a large data frame with 793 rows and 17 columns. The columns are labeled with numerical values ranging from 3946 to 3562. At the bottom of the console, there is an error message:

```
[541] 3946 4188 4318 5350 5788 12420 13262 19602 23976 24542 16554 18720
[553] 20829 21304 25295 25777 25830 26511 29925 31471 11229 12314 12772 17417
[565] 21132 21211 24992 25557 26028 27721 1592 2189 17959 20751 21974 22637
[577] 23989 25598 27800 29325 9185 10915 13678 16193 21418 23886 24960 26126
[589] 27199 36480 13145 17830 19081 19225 20618 21891 23931 24318 25285 26097
[601] 18096 18391 19465 19531 19894 22779 23062 24239 27887 33374 12303 16078
[613] 17053 18206 21307 21391 21461 27438 31424 31791 2973 4463 8754 9150
[625] 12610 14304 21688 23217 23323 24270 10340 12251 17711 19112 20925 21208
[637] 21661 23553 24559 35157 3907 16068 16688 19569 21266 21433 26345 27674
[649] 27686 50387 3828 17381 17590 18930 22479 23635 25049 27558 31655 32477
[661] 10003 10014 14281 17319 20208 21160 22272 22814 27015 27600 12090 13162
[673] 16088 18195 21167 21293 24031 25420 27051 48991 3867 5144 10800 11273
[685] 14568 22189 22328 22598 26400 34175 1676 7924 9795 15087 22052 23143
[697] 25247 26276 28598 38717 16956 16975 20015 20103 22105 22298 23426 25380
[709] 26503 31773 6409 14278 15050 16225 17317 18187 18805 21960 36557 40719
[721] 8017 14613 19854 22021 22590 22881 23055 23204 24738 31344 17673 18464
[733] 19830 20685 20907 21545 21822 25357 26786 34269 4836 17431 21616 25218
[745] 2879 8507 10555 12920 14376 16951 17656 19455 21020 22255 23645 24349
[757] 25981 26034 26236 26788 26986 27231 30108 38958 1737 2392 11836 12412
[769] 12878 13318 14938 16163 17139 18257 18511 19101 19500 19853 21770 21877
[781] 22758 22826 23388 24069 24664 24740 26328 28298 28398 29707 32458 32619
[793] 33962 34998 5826 7755 10102 10986 14242 16229 19095 20484 25979 35662
> Mileage
Error: object 'Mileage' not found
> |
```

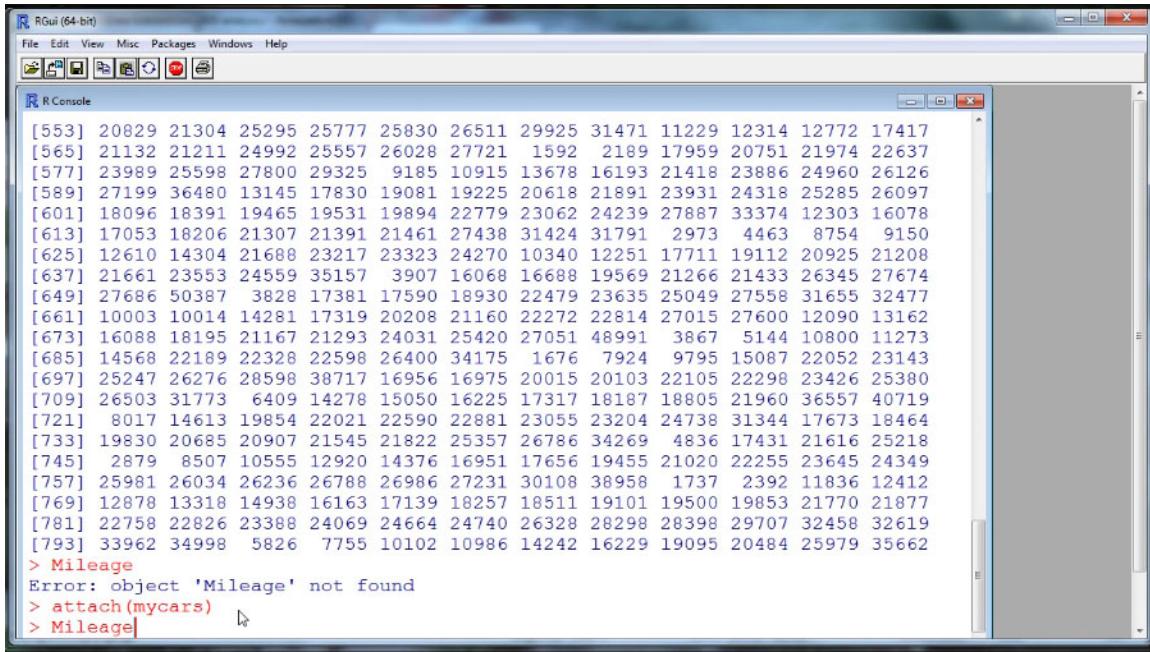
And there we have the mileages of all the cars in our data set.

But if we want to look at one column by name, say by typing mileage, we get an error message. R doesn't know what we're talking about.



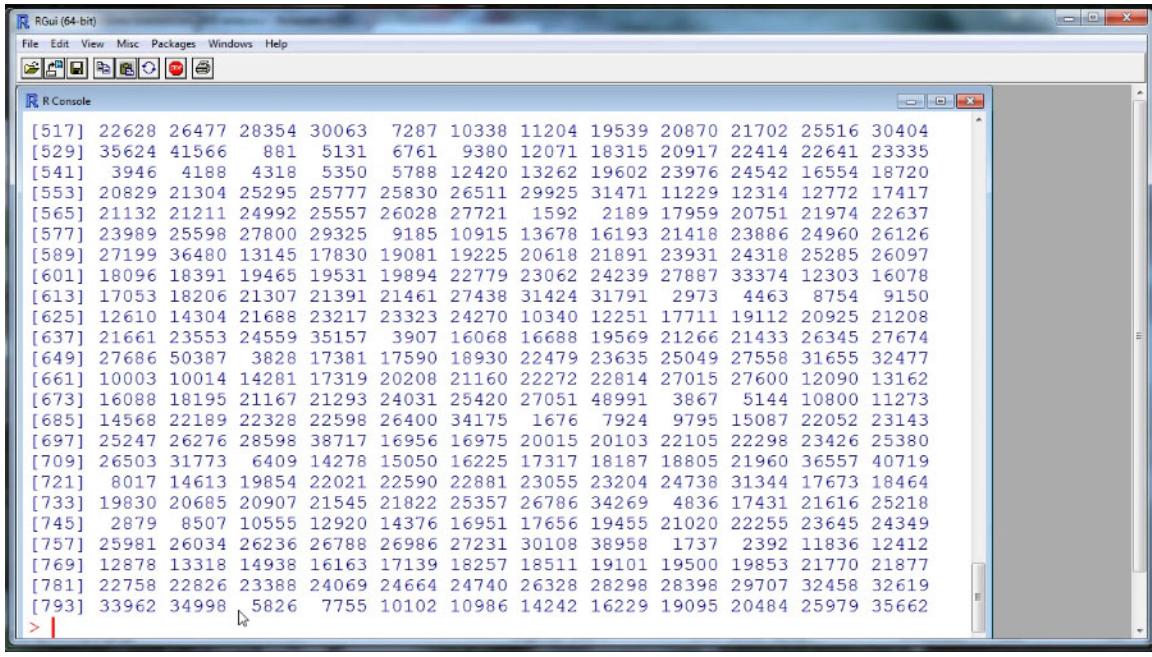
```
[541] 3946 4188 4318 5350 5788 12420 13262 19602 23976 24542 16554 18720
[553] 20829 21304 25295 25777 25830 26511 29925 31471 11229 12314 12772 17417
[565] 21132 21211 24992 25557 26028 27721 1592 2189 17959 20751 21974 22637
[577] 23989 25598 27800 29325 9185 10915 13678 16193 21418 23886 24960 26126
[589] 27199 36480 13145 17830 19081 19225 20618 21891 23931 24318 25285 26097
[601] 18096 18391 19465 19531 19894 22779 23062 24239 27887 33374 12303 16078
[613] 17053 18206 21307 21391 21461 27438 31424 31791 2973 4463 8754 9150
[625] 12610 14304 21688 23217 23323 24270 10340 12251 17711 19112 20925 21208
[637] 21661 23553 24559 35157 3907 16068 16688 19569 21266 21433 26345 27674
[649] 27686 50387 3828 17381 17590 18930 22479 23635 25049 27558 31655 32477
[661] 10003 10014 14281 17319 20208 21160 22272 22814 27015 27600 12090 13162
[673] 16088 18195 21167 21293 24031 25420 27051 48991 3867 5144 10800 11273
[685] 14568 22189 22328 22598 26400 34175 1676 7924 9795 15087 22052 23143
[697] 25247 26276 28598 38717 16956 16975 20015 20103 22105 22298 23426 25380
[709] 26503 31773 6409 14278 15050 16225 17317 18187 18805 21960 36557 40719
[721] 8017 14613 19854 22021 22590 22881 23055 23204 24738 31344 17673 18464
[733] 19830 20685 20907 21545 21822 25357 26786 34269 4836 17431 21616 25218
[745] 2879 8507 10555 12920 14376 16951 17656 19455 21020 22255 23645 24349
[757] 25981 26034 26236 26788 26986 27231 30108 38958 1737 2392 11836 12412
[769] 12878 13318 14938 16163 17139 18257 18511 19101 19500 19853 21770 21877
[781] 22758 22826 23388 24069 24664 24740 26328 28298 28398 29707 32458 32619
[793] 33962 34998 5826 7755 10102 10986 14242 16229 19095 20484 25979 35662
> Mileage
Error: object 'Mileage' not found
> Mileage = mycars[,2]
```

It will be easier to work with the data if we create a variable for each column of the data. We could do this for each column separately. For example, mileage equals mycars, square bracket, comma 2.



```
RGui (64-bit)
File Edit View Misc Packages Windows Help
R Console
[553] 20829 21304 25295 25777 25830 26511 29925 31471 11229 12314 12772 17417
[565] 21132 21211 24992 25557 26028 27721 1592 2189 17959 20751 21974 22637
[577] 23989 25598 27800 29325 9185 10915 13678 16193 21418 23886 24960 26126
[589] 27199 36480 13145 17830 19081 19225 20618 21891 23931 24318 25285 26097
[601] 18096 18391 19465 19531 19894 22779 23062 24239 27887 33374 12303 16078
[613] 17053 18206 21307 21391 21461 27438 31424 31791 2973 4463 8754 9150
[625] 12610 14304 21688 23217 23323 24270 10340 12251 17711 19112 20925 21208
[637] 21661 23553 24559 35157 3907 16068 16688 19569 21266 21433 26345 27674
[649] 27686 50387 3828 17381 17590 18930 22479 23635 25049 27558 31655 32477
[661] 10003 10014 14281 17319 20208 21160 22272 22814 27015 27600 12090 13162
[673] 16088 18195 21167 21293 24031 25420 27051 48991 3867 5144 10800 11273
[685] 14568 22189 22328 22598 26400 34175 1676 7924 9795 15087 22052 23143
[697] 25247 26276 28598 38717 16956 16975 20015 20103 22105 22298 23426 25380
[709] 26503 31773 6409 14278 15050 16225 17317 18187 18805 21960 36557 40719
[721] 8017 14613 19854 22021 22590 22881 23055 23204 24738 31344 17673 18464
[733] 19830 20685 20907 21545 21822 25357 26786 34269 4836 17431 21616 25218
[745] 2879 8507 10555 12920 14376 16951 17656 19455 21020 22255 23645 24349
[757] 25981 26034 26236 26788 26986 27231 30108 38958 1737 2392 11836 12412
[769] 12878 13318 14938 16163 17139 18257 18511 19101 19500 19853 21770 21877
[781] 22758 22826 23388 24069 24664 24740 26328 28298 28398 29707 32458 32619
[793] 33962 34998 5826 7755 10102 10986 14242 16229 19095 20484 25979 35662
> Mileage
Error: object 'Mileage' not found
> attach(mycars)
> Mileage|
```

But it's faster to do this automatically. In R this is called attaching the data. We just type attach, and then the variable name where we stored the data set. Now if we type the variable mileage...



The screenshot shows the RGui (64-bit) application window. The title bar reads "RGui (64-bit)". The menu bar includes "File", "Edit", "View", "Misc", "Packages", "Windows", and "Help". Below the menu is a toolbar with various icons. The main area is titled "R Console". It displays a long vector of numerical values representing mileages, starting with [517] 22628 and ending with [793] 33962. The vector contains approximately 2800 elements. The console window has scroll bars on the right and bottom.

```
[517] 22628 26477 28354 30063 7287 10338 11204 19539 20870 21702 25516 30404  
[529] 35624 41566 881 5131 6761 9380 12071 18315 20917 22414 22641 23335  
[541] 3946 4188 4318 5350 5788 12420 13262 19602 23976 24542 16554 18720  
[553] 20829 21304 25295 25777 25830 26511 29925 31471 11229 12314 12772 17417  
[565] 21132 21211 24992 25557 26028 27721 1592 2189 17959 20751 21974 22637  
[577] 23989 25598 27800 29325 9185 10915 13678 16193 21418 23886 24960 26126  
[589] 27199 36480 13145 17830 19081 19225 20618 21891 23931 24318 25285 26097  
[601] 18096 18391 19465 19531 19894 22779 23062 24239 27887 33374 12303 16078  
[613] 17053 18206 21307 21391 21461 27438 31424 31791 2973 4463 8754 9150  
[625] 12610 14304 21688 23217 23323 24270 10340 12251 17711 19112 20925 21208  
[637] 21661 23553 24559 35157 3907 16068 16688 19569 21266 21433 26345 27674  
[649] 27686 50387 3828 17381 17590 18930 22479 23635 25049 27558 31655 32477  
[661] 10003 10014 14281 17319 20208 21160 22272 22814 27015 27600 12090 13162  
[673] 16088 18195 21167 21293 24031 25420 27051 48991 3867 5144 10800 11273  
[685] 14568 22189 22328 22598 26400 34175 1676 7924 9795 15087 22052 23143  
[697] 25247 26276 28598 38717 16956 16975 20015 20103 22105 22298 23426 25380  
[709] 26503 31773 6409 14278 15050 16225 17317 18187 18805 21960 36557 40719  
[721] 8017 14613 19854 22021 22590 22881 23055 23204 24738 31344 17673 18464  
[733] 19830 20685 20907 21545 21822 25357 26786 34269 4836 17431 21616 25218  
[745] 2879 8507 10555 12920 14376 16951 17656 19455 21020 22255 23645 24349  
[757] 25981 26034 26236 26788 26986 27231 30108 38958 1737 2392 11836 12412  
[769] 12878 13318 14938 16163 17139 18257 18511 19101 19500 19853 21770 21877  
[781] 22758 22826 23388 24069 24664 24740 26328 28298 28398 29707 32458 32619  
[793] 33962 34998 5826 7755 10102 10986 14242 16229 19095 20484 25979 35662
```

...we get back that long vector of mileages that we saw when we were looking at the second column of data.

Control Flow sample problem:
Analyzing a data set in R

Part 2

Let's look at the second part of our hypothetical control flow problem.

b. Print the types (hatchback, sedan, etc.) of all the cars with prices less than \$10,000.

We want to print the types of all the cars with prices less than \$10,000. There are several ways to solve this problem, but we're learning about control flow, so let's do it that way.

Outline a plan

- **Pseudocode:** Natural language description of how the program will work

When you're writing a computer program it's a good idea to plan out how you're going to write it using pseudo code, which is a description of how the program will work in plain language.

b. Print the types (hatchback, sedan, etc.) of all the cars with prices less than \$10,000.

- Find all the cars with prices less than \$10,000.
 - Look at each row (or car) in the data.
 - Check whether that car's price is less than \$10,000.
- Print what type they are.

So what do we want our program to do? Well, first we want to find all of the cars with prices less than \$10,000, and then we want to print what type they are. Let's see if we can break down that first bullet point into a little more detail. In order to find the cars with prices less than \$10,000. We can first look at each row, or car in the data, and check whether its price is less than \$10,000.

Recognizing Control Flow

- "Each", "every", "all" → **For or while loop**
 - "Look at each row in the data."
- Number of steps/iterations is known at the start → **For loop**
 - 1 step per row (or car)

So how can we write this using a control flow structure? Well we use the words, look at each row in the data. The word each is a good clue that we either want a for loop or a while loop. In this case we know how many steps or iterations we want to do. One step per row, and we know how many rows the data set has, we can look at the start of the program. That's a good clue that a for loop will be helpful. We could also do this using a while loop, but a for loop is more intuitive.

Our Pseudocode

```
for( each car ){
    Check whether the price is less than $10,000
    If so, print the type
}
```

- "Whether", "if", "unless" → if statement

So we can begin our pseudocode with a for loop. For each car we're going to do some things. So what things do we want to do for each car? We said we wanted to check whether its price was less than \$10,000, and if so, print its type. Here we've used the words whether and if. Those are good hints that we want to use an if statement.

Our Pseudocode

```
for( each car ){
    if(This car's price < $10,000){
        print this car's type
    }
}
```

So we can fill that in with pseudocode by saying, if this car's price is less than \$10,000. Notice that we're still using plain English to say this car's price, and each car. We'll fill those in with correct R code later on. Right now we're just getting our general understanding of the structure of the code. Now inside this if statement we want to print this car's type, and we can fill in the correct R code to do this later as well. For now, how can we be more specific about what goes inside the parentheses in the for loop?

Indexing the for loop

- `for(object in vector){ }`
- **object** is a new variable name
 - Ex: `current_car`
 - Created by this line of code
- **vector** is a set of items

The general structure for for loops in R is for some object in some vector. Where object is the name of a new variable. For example, in this case we could use `current_underscore car`. This variable is created by this line of code. So unlike in some other programming languages we don't need to initialize or create a variable before the for loop starts. Vector is a set of items that object is going to iterate over. That is, object will start by being equal to the first element in vector. The second time through the for loop it will be equal to the second element, and so on.

What happens inside the **for** loop?

- Working with a single variable/type of item → Let **vector** be that variable
- Example:

```
code  friends = c("Caitlin", "Jonathan", "Gwen")
      for(current_person in friends) {
        print( paste(current_person, "is my friend." ) )
      }
```

```
output [1] "Caitlin is my friend."
[1] "Jonathan is my friend."
[1] "Gwen is my friend."
```

In order to decide what goes in that variable vector, we need to think about what happens inside the for loop. If we're working with a single variable, or a single type of item, then we can let vector be that variable. For example, suppose I want to print the sentence, current person is my friend. Where current person gets filled in with the name of each of my friends, from this vector friends. That means that my for loop can iterate over that vector, friends. Because I'm only working with that one variable of friends. In this case this code would produce output that looks like this.

What happens inside the for loop?

- Working with multiple variables/columns of data → Let `vector` be a set of numbers
- Usually integers from 1 to # of rows (or number of elements of each variable)
- *object* will index which row/person/etc. you're currently examining

On the other hand, if we're working with multiple variables, or multiple columns of data, then it's a good idea to let vector be a set of numbers, usually the integers from 1 up to the number of rows or the number of elements of each variable. Then object will index which row or person or whatever we're currently examining.

Example

```
code friends = c("Caitlin", "Jonathan", "Gwen")
siblings = c(1, 0, 2)
for(index in 1:3){
  print(paste(friends[index], "has", siblings[index], "sibling(s)."))
}

output [1] "Caitlin has 1 sibling(s)."
[1] "Jonathan has 0 sibling(s)."
[1] "Gwen has 2 sibling(s)."
```

For example, suppose I want to print sentences telling how many siblings each of my friends has. Here I'm working with two vectors. Friends, which gives my friends names, and siblings, which gives the number of siblings each friend has. Because I'm working with two vectors in the for loop I'm letting vector be the numbers from one to three.

The first time through this for loop the variable index will equal one. So I'll be printing friends square bracket one. That means the first element of the friends vector. So that's Caitlin's name. Then I'll print the word has, the first element of the siblings vector, which is the number one, and then the word sibling. And we'll repeat this for each of the values of index from one to three. In this case this code produces output that looks like this.

 SELF-ASSESSMENT

What variable would you iterate over for the cars problem?

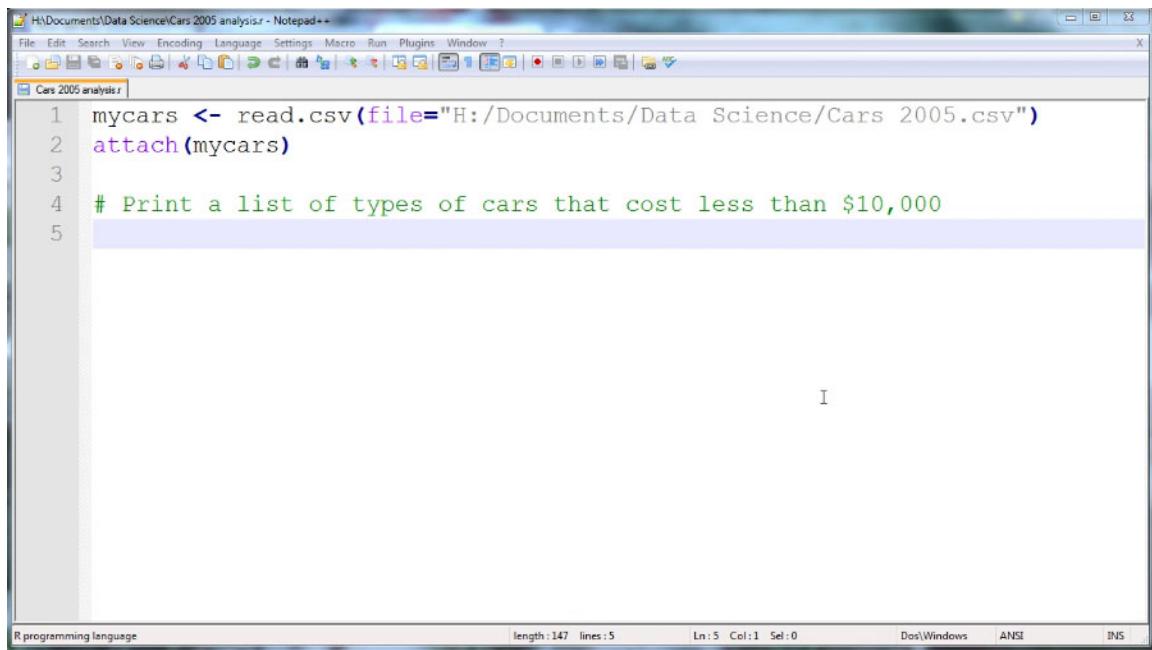
- Price
- Type
- Integers from 1 to the number of cars.

SUBMIT

Question 1: What variable would you iterate over for the cars problem?

- Price
- Type
- Integers from 1 to the number of cars.

*See page 62 for Question 1 answer.



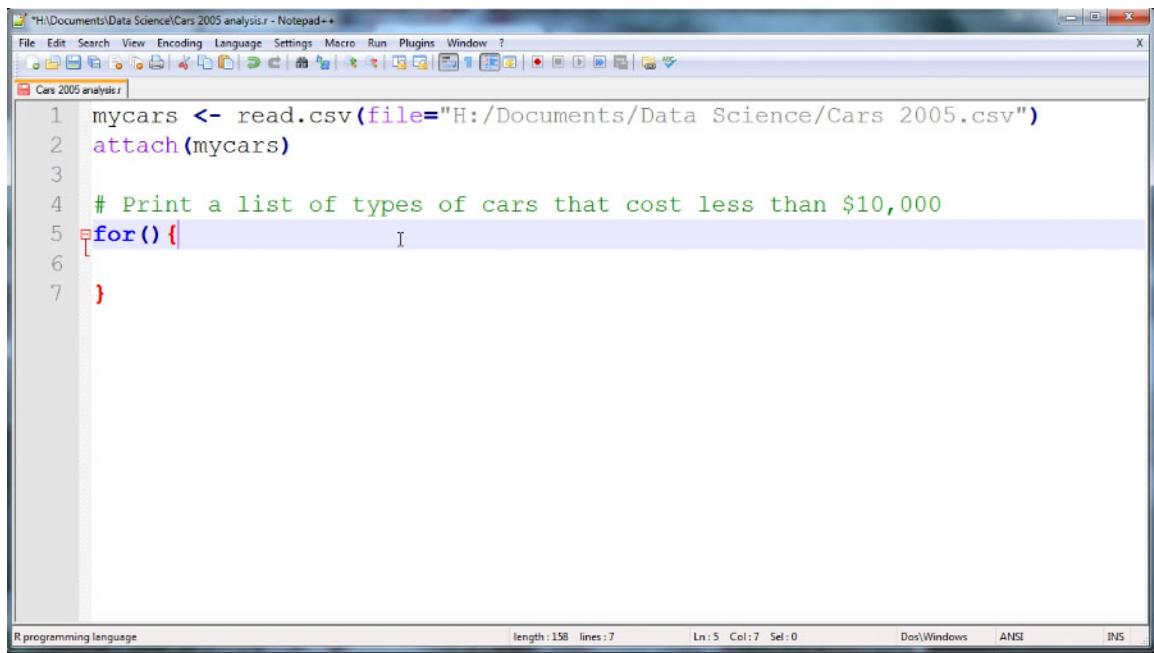
A screenshot of the Notepad++ text editor. The title bar reads "H:\Documents\Data Science\Cars 2005 analysis.r - Notepad++". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, Window, and ?.

The code in the editor is:

```
1 mycars <- read.csv(file="H:/Documents/Data Science/Cars 2005.csv")
2 attach(mycars)
3
4 # Print a list of types of cars that cost less than $10,000
5
```

The status bar at the bottom shows "R programming language", "length:147 lines:5", "Ln:5 Col:1 Sel:0", "Dos\Windows", "ANSI", and "INS".

Let's write the code to print a list of the types of cars that cost less than \$10,000. Whenever you're working with control flow structures it's a good idea to work in a script. So you have everything the way you want it before you try to run the program.

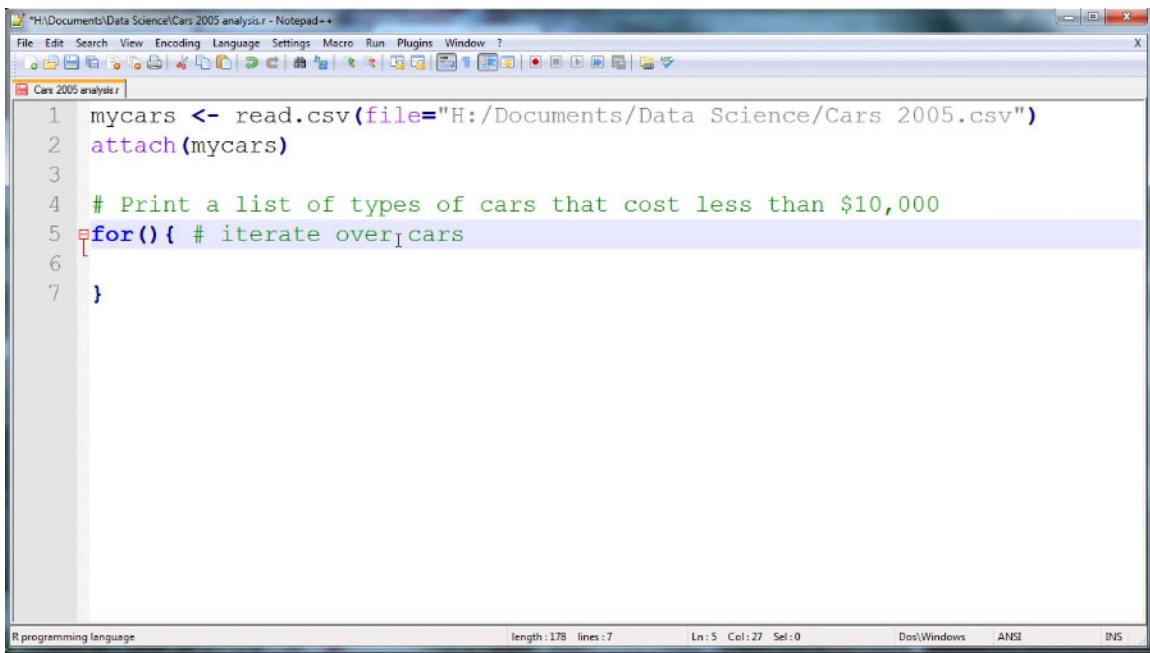


The screenshot shows a Notepad++ window titled "H:\Documents\Data Science\Cars 2005 analysis.r - Notepad++". The code in the editor is:

```
1 mycars <- read.csv(file="H:/Documents/Data Science/Cars 2005.csv")
2 attach(mycars)
3
4 # Print a list of types of cars that cost less than $10,000
5 for() {
6
7 }
```

The status bar at the bottom indicates: R programming language, length: 158, lines: 7, Ln: 5 Col: 7 Sel: 0, Dos\Windows, ANSI, INS.

Based on our pseudocode the outermost structure should be a for loop. So I'll type that structure in here.

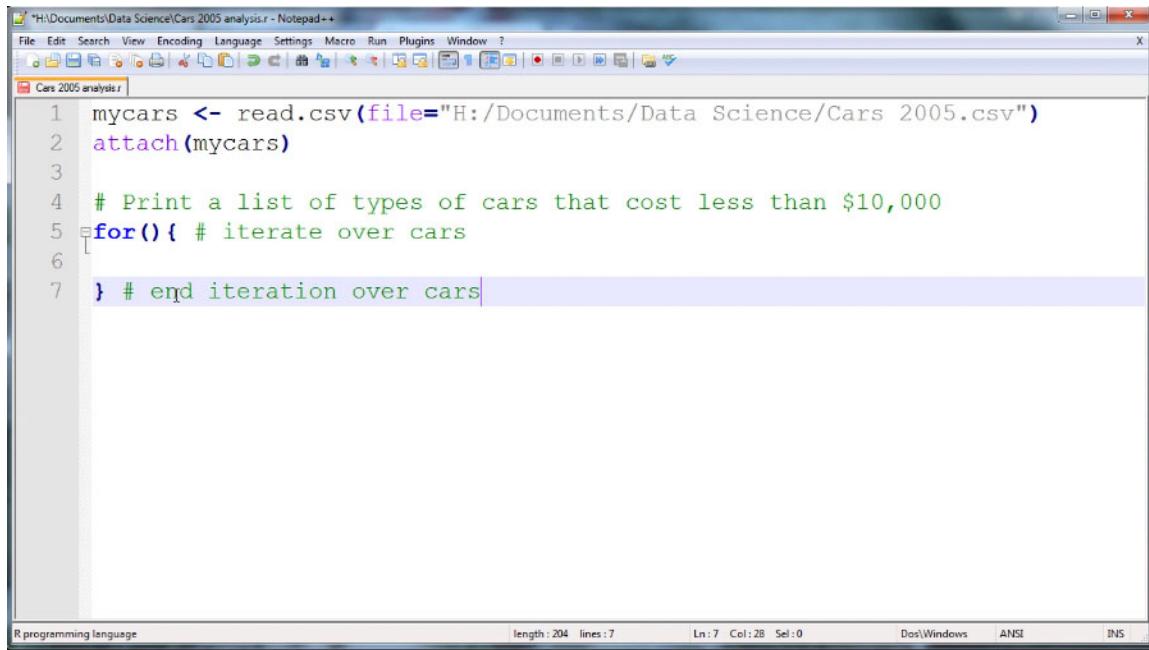


The screenshot shows a Notepad++ window titled "H:\Documents\Data Science\Cars 2005 analysis.r - Notepad++". The code in the editor is as follows:

```
1 mycars <- read.csv(file="H:/Documents/Data Science/Cars 2005.csv")
2 attach(mycars)
3
4 # Print a list of types of cars that cost less than $10,000
5 for() { # iterate over cars
6
7 }
```

The status bar at the bottom indicates "R programming language", "length : 178 lines : 7", "Ln : 5 Col : 27 Sel : 0", "Dos\Windows", "ANSI", and "INS".

And I'll also label what this for loop is supposed to do using a comment. This for loop should iterate over the cars. Remember R will ignore everything that comes after the pound sign or hash mark on a line. So using a comment this way is a good way to remind yourself and other programmers what this code is supposed to do.



A screenshot of the Notepad++ text editor. The title bar reads "H:\Documents\Data Science\Cars 2005 analysis.r - Notepad++". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, Window, and ?.

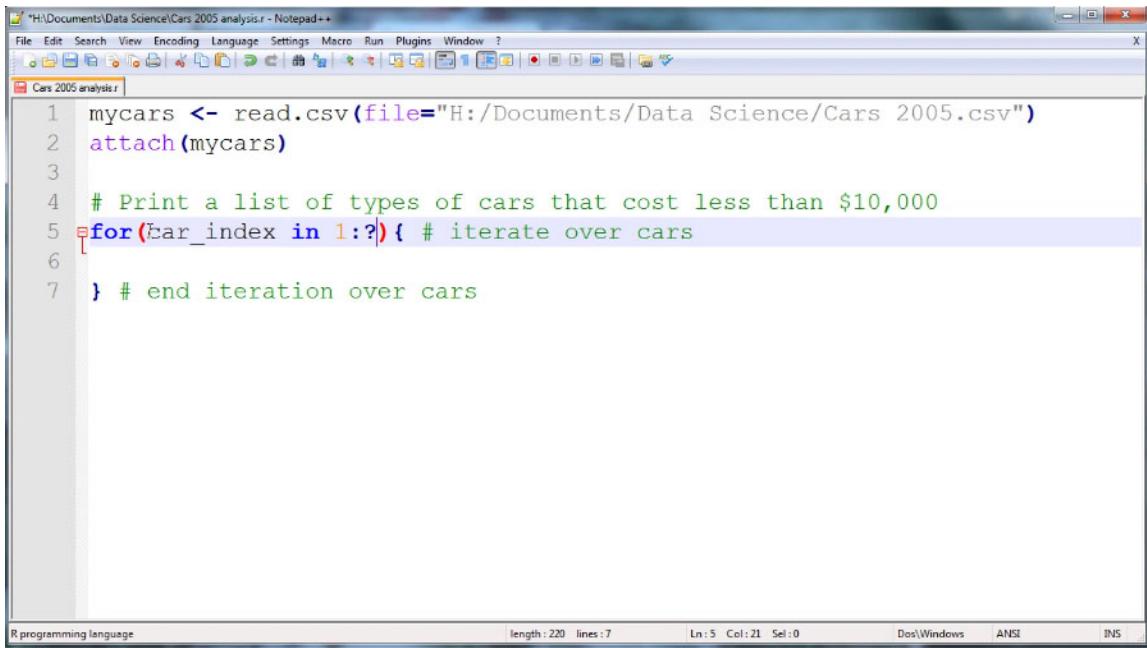
The code in the editor is:

```
1 mycars <- read.csv(file="H:/Documents/Data Science/Cars 2005.csv")
2 attach(mycars)
3
4 # Print a list of types of cars that cost less than $10,000
5 for() { # iterate over cars
6
7 } # end iteration over cars
```

The status bar at the bottom shows "R programming language", "length : 204 lines : 7", "Ln : 7 Col : 28 Sel : 0", "Dos\Windows", "ANSI", and "INS".

I also like to put a comment at the end of my for loop, to remind myself what this for loop was doing.

Here we're ending the iteration over the cars. This is helpful for when I have multiple control flow structures nested together. I can keep track of which closed curly bracket goes with which structure.



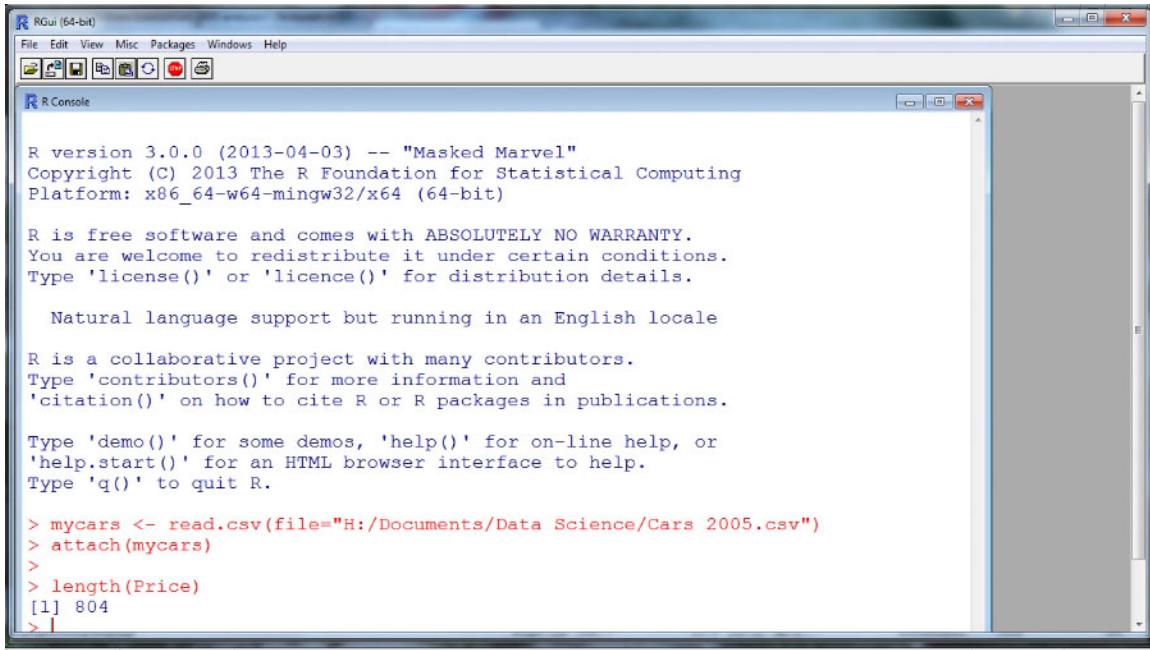
The screenshot shows a Notepad++ window titled "Cars 2005 analysis.r". The code in the editor is:

```
1 mycars <- read.csv(file="H:/Documents/Data Science/Cars 2005.csv")
2 attach(mycars)
3
4 # Print a list of types of cars that cost less than $10,000
5 for(car_index in 1:?) { # iterate over cars
6
7 } # end iteration over cars
```

The status bar at the bottom indicates: length : 220 lines : 7 Ln : 5 Col : 21 Sel : 0 Dos\Windows ANSI INS.

In this loop we'll be testing the price and printing the type. That's two different columns of data, so we want to use a numerical index. I'll call it car underscore index. And we want that to iterate over the numbers from one up to some number. We want to iterate over all the cars.

So the maximum value that car index can take on, should be the total number of rows in the data set. That's the same as the length of any one of the column vectors.



```
R version 3.0.0 (2013-04-03) -- "Masked Marvel"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

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

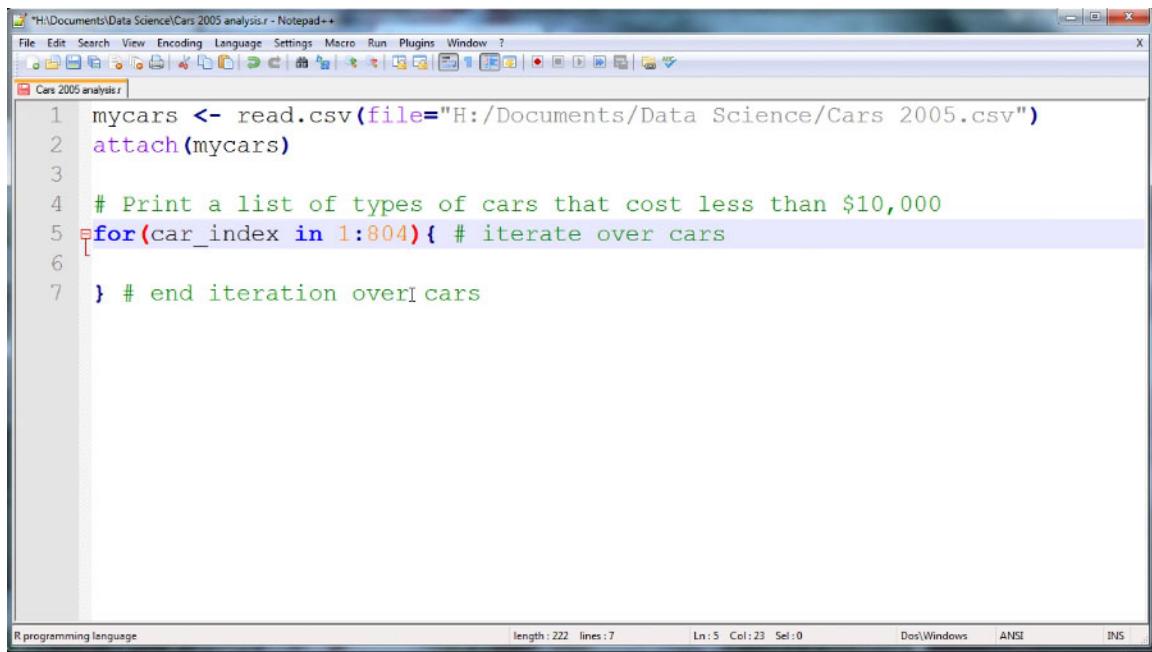
Natural language support but running in an English locale

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

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

> mycars <- read.csv(file="H:/Documents/Data Science/Cars 2005.csv")
> attach(mycars)
>
> length(Price)
[1] 804
> |
```

If we switch over to R we can use the length function to find the length of any of those column vectors, say price. And there are 804 entries in that data set.



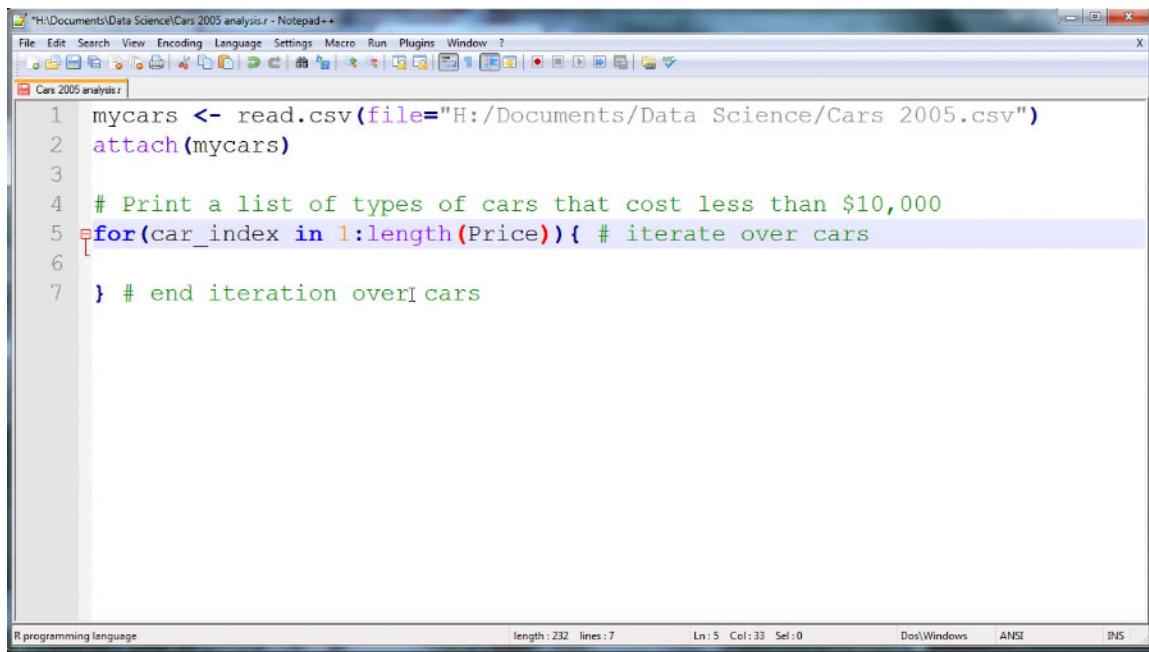
A screenshot of the Notepad++ text editor. The title bar reads "H:\Documents\Data Science\Cars 2005 analysis.r - Notepad++". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, Window, and ?.

The code in the editor is:

```
1 mycars <- read.csv(file="H:/Documents/Data Science/Cars 2005.csv")
2 attach(mycars)
3
4 # Print a list of types of cars that cost less than $10,000
5 for(car_index in 1:804){ # iterate over cars
6
7 } # end iteration over cars
```

The status bar at the bottom shows "R programming language", "length : 222 lines : 7", "Ln : 5 Col : 23 Sel : 0", "Dos\Windows", "ANSI", and "INS".

So we could type 804 in place of the question mark.



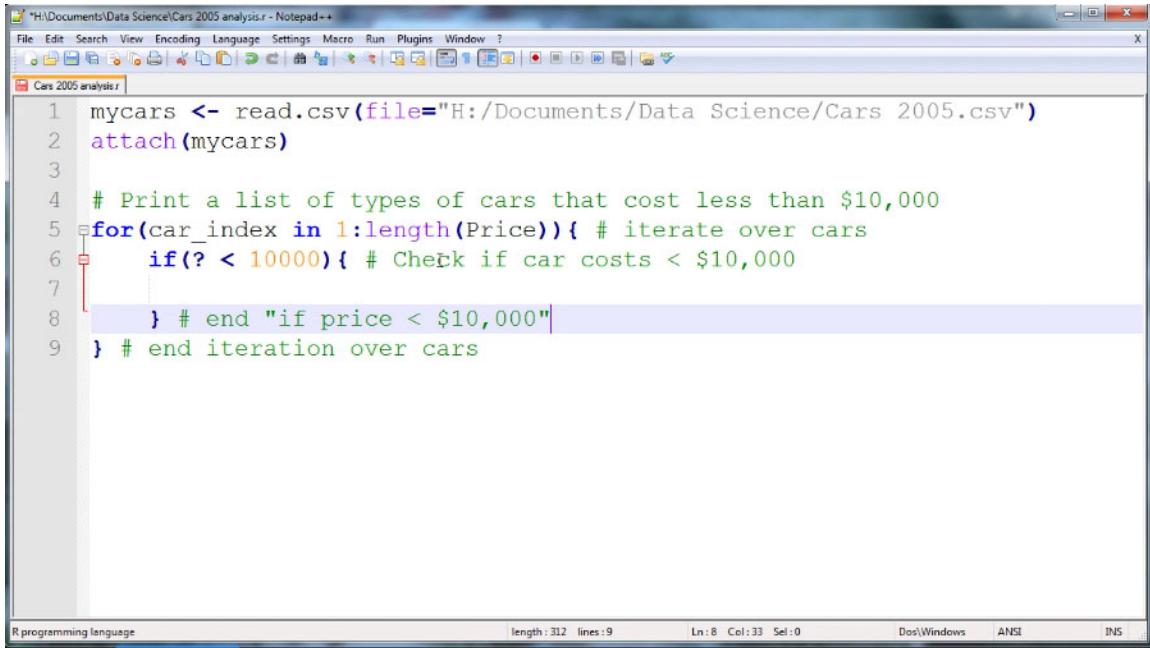
A screenshot of the Notepad++ text editor. The title bar reads "H:\Documents\Data Science\Cars 2005 analysis.r - Notepad++". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, Window, and ?.

The code in the editor is:

```
1 mycars <- read.csv(file="H:/Documents/Data Science/Cars 2005.csv")
2 attach(mycars)
3
4 # Print a list of types of cars that cost less than $10,000
5 for(car_index in 1:length(Price)){ # iterate over cars
6
7 } # end iteration over cars
```

The status bar at the bottom shows "R programming language", "length : 232 lines : 7", "Ln : 5 Col : 33 Sel : 0", "Dos\Windows", "ANSI", and "INS".

But to make our code more general, we could use the length of price instead. That way if we got a new data set of used cars in some other year, all we would have to do is change the file name that we're reading in, and the rest of our code, including this for loop, would still work.

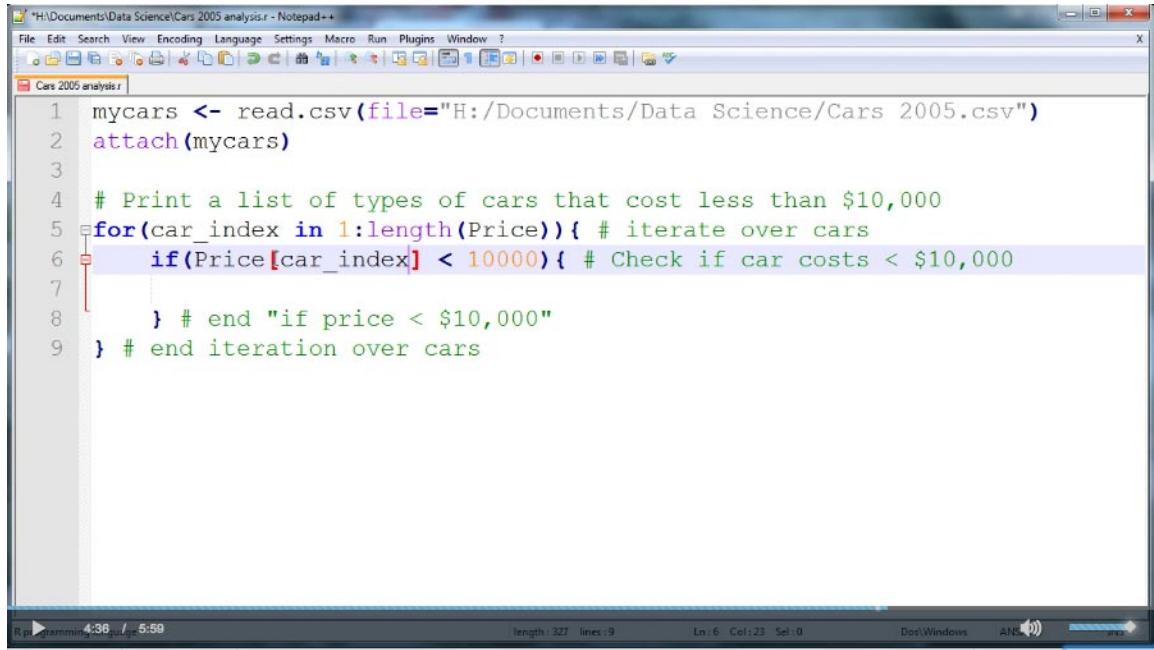


The screenshot shows a Notepad++ window titled "H:\Documents\Data Science\Cars 2005 analysis.r - Notepad++". The code is written in R and reads a CSV file named "Cars 2005.csv" into a dataset called "mycars". It then iterates over the rows of the dataset, checking if the price of each car is less than \$10,000. The code is as follows:

```
1 mycars <- read.csv(file="H:/Documents/Data Science/Cars 2005.csv")
2 attach(mycars)
3
4 # Print a list of types of cars that cost less than $10,000
5 for(car_index in 1:length(Price)){ # iterate over cars
6   if(? < 10000){ # Check if car costs < $10,000
7     }
8   } # end "if price < $10,000"
9 } # end iteration over cars
```

Now inside the for loop we want to test if the car's price is less than \$10,000.

I'm just going to leave a question mark for the variable that we're testing, but we want to test if it's less than \$10,000. And I'll label this with comments as well. Remember that the variable car index tells us which car, or row, we're looking at. It starts at one in the first iteration through the loop, and increases to two, three, and so on, up to 804.

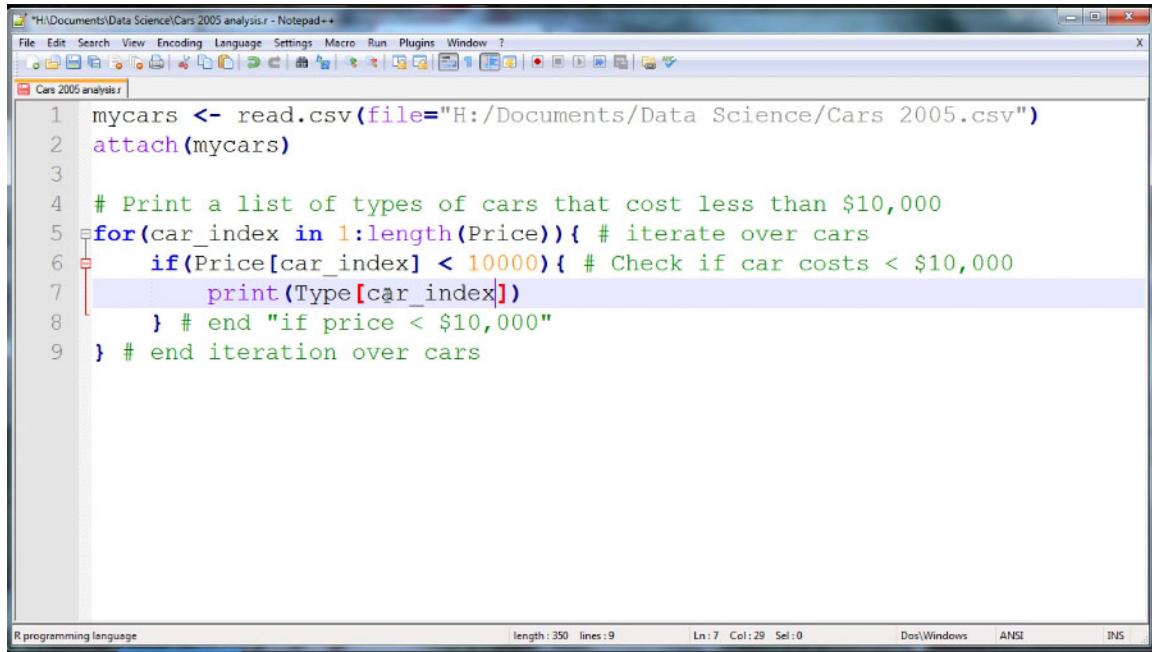


The screenshot shows a Notepad++ window titled "H:\Documents\Data Science\Cars 2005 analysis.r - Notepad++". The code in the editor is as follows:

```
1 mycars <- read.csv(file="H:/Documents/Data Science/Cars 2005.csv")
2 attach(mycars)
3
4 # Print a list of types of cars that cost less than $10,000
5 for(car_index in 1:length(Price)){ # iterate over cars
6   if(Price[car_index] < 10000){ # Check if car costs < $10,000
7     }
8   } # end "if price < $10,000"
9 } # end iteration over cars
```

The status bar at the bottom of the Notepad++ window displays: R programming 4:36 / 5:59, length: 327, lines: 9, Ln: 6 Col: 23 Sel: 0, Dos\Windows, ANSI.

So to look at the price of the current car we want to look at the price column, or variable, and the car index location. So on the first time through the loop car index equals one, we'll be looking at price square bracket one, or the first element in the price vector.

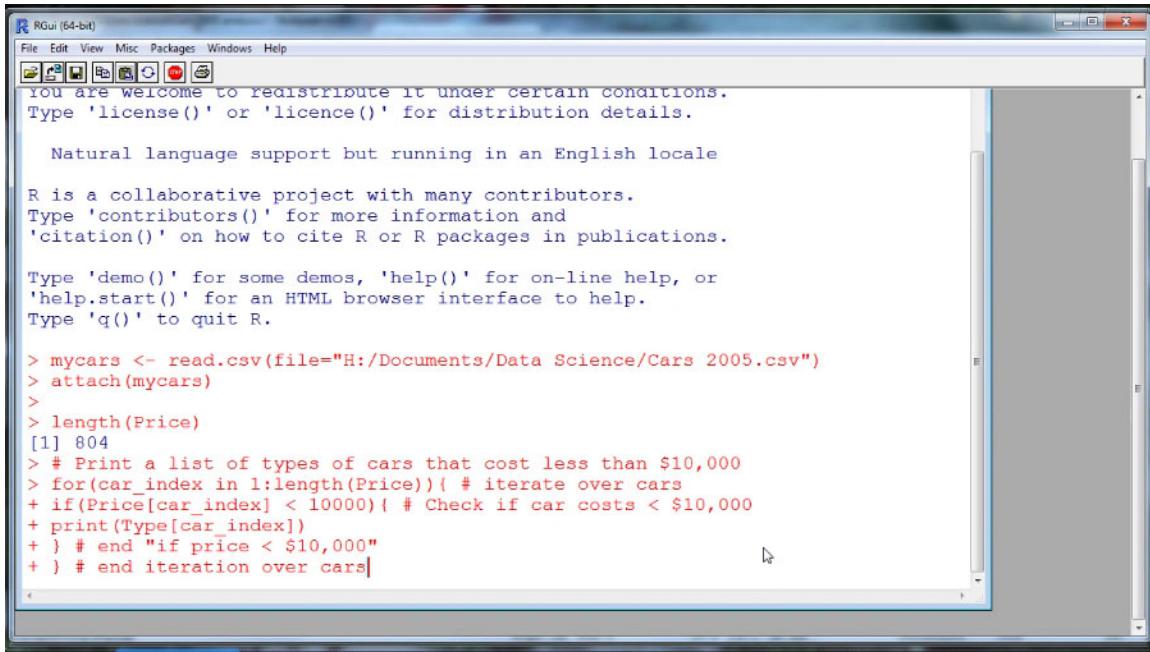


The screenshot shows a Notepad++ window titled "H:\Documents\Data Science\Cars 2005 analysis.r - Notepad++". The code in the editor is as follows:

```
1 mycars <- read.csv(file="H:/Documents/Data Science/Cars 2005.csv")
2 attach(mycars)
3
4 # Print a list of types of cars that cost less than $10,000
5 for(car_index in 1:length(Price)){ # iterate over cars
6   if(Price[car_index] < 10000){ # Check if car costs < $10,000
7     print(Type[car_index])
8   } # end "if price < $10,000"
9 } # end iteration over cars
```

The Notepad++ status bar at the bottom indicates: R programming language, length: 350, lines: 9, Ln: 7, Col: 29, Sel: 0, Dos\Windows, ANSI, INS.

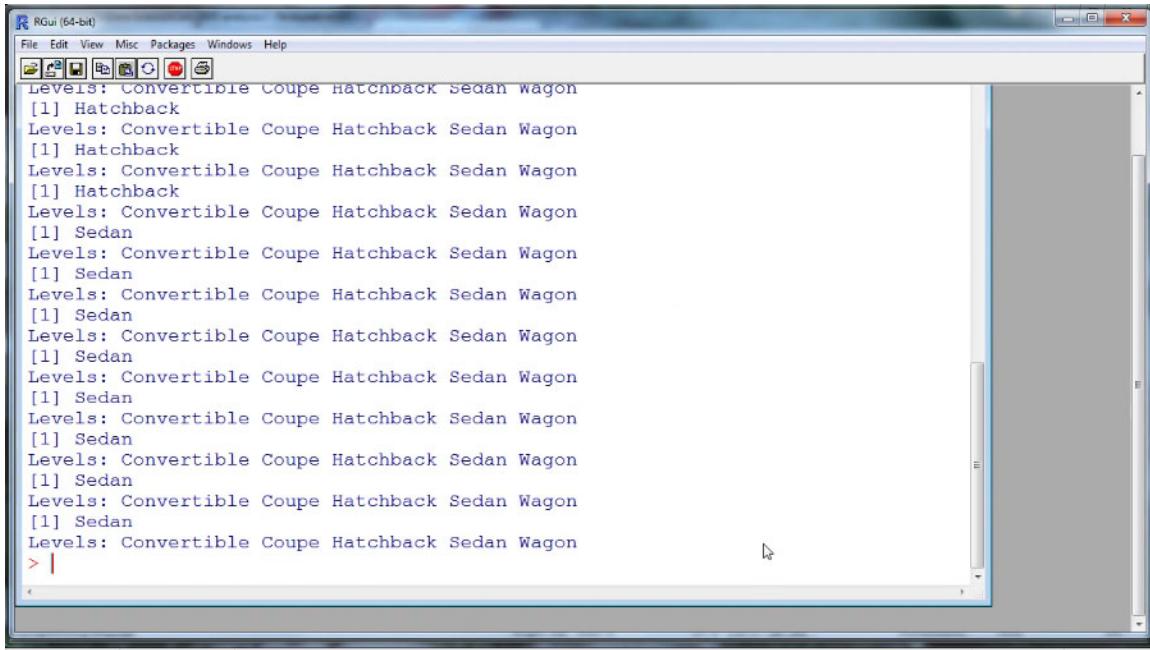
Finally, if this inequality is true, then we want to print the type of the current car using the print function. And the variable we want to print is the type of that same car that we're looking at. So car index.



The screenshot shows the RGui (64-bit) application window. The menu bar includes File, Edit, View, Misc, Packages, Windows, and Help. Below the menu is a toolbar with icons for file operations. The main area displays the R startup message and a command-line session:

```
you are welcome to redistribute it under certain conditions.  
Type 'license()' or 'licence()' for distribution details.  
  
Natural language support but running in an English locale  
  
R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.  
  
Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.  
  
> mycars <- read.csv(file="H:/Documents/Data Science/Cars 2005.csv")  
> attach(mycars)  
>  
> length(Price)  
[1] 804  
> # Print a list of types of cars that cost less than $10,000  
> for(car_index in 1:length(Price)){ # iterate over cars  
+ if(Price[car_index] < 10000){ # Check if car costs < $10,000  
+ print(Type[car_index])  
+ } # end "if price < $10,000"  
+ } # end iteration over cars|
```

Now we think our code is complete, so we can copy and paste it into R.

A screenshot of the RGui (64-bit) application window. The menu bar includes File, Edit, View, Misc, Packages, Windows, and Help. Below the menu is a toolbar with icons for file operations. The main area displays R code and its output. The code consists of several identical lines: "Levels: Convertible Coupe Hatchback Sedan Wagon [1] Hatchback". This indicates that the variable "type" has four levels: Convertible, Coupe, Hatchback, and Sedan, but only Hatchback and Sedan appear in the subset of the dataset where the price is less than \$10,000.

Here we see that the first car in the data set that costs less than \$10,000 was a hatchback. The second was us sedan, and so on. The levels here are the different values that the type variable could take on. This was a categorical variable that could take on the values of convertible, coupe, hatchback, sedan, or wagon. But as it turns out, only hatchbacks and sedans appear in the group that costs less than \$10,000.

Control Flow sample problem:
Analyzing a data set in R

Part 3

Let's look at the third part of our hypothetical control flow problem.

c. Count the number of cars with prices less than \$10,000.

```
pseudocode for( each car ){
    if(This car's price < $10,000){
        count it
    }
}
```

Here we want to count the number of cars with prices less than \$10,000. We already printed the types of each of these cars, and there weren't very many of them. So we could just count them by hand, but we'd like to have a way to automate this process in case someone gave us a larger data set. So to do this let's start with the pseudocode that we wrote for part B. For each car we check if the car's price is less than \$10,000, but in this case we don't want to print the car's type, we want to count it.

Updating a numerical quantity throughout a for loop

- Keep track of the number of cars with prices less than \$10,000 that we've seen *so far*
- Need a new variable, say, `cheap_count`
- Each time we see a cheap car, add one to this variable:

```
cheap_count = cheap_count + 1
```

So how can we count it? Well, we want to keep track of the number of cars with prices less than \$10,000 that we've seen so far. That means we need a new variable. In this case, we could use `cheap_underscore_count`. Then each time we see a cheap car, we would add one to this variable, using the equation `cheap count` equals `cheap count plus one`.

Notice that the left hand side of this equation has a variable that's being assigned a new value. The right hand side has the value that's being assigned to that variable. So even though this is an equation, it's not like an equation for math class. We're not saying that `x` equals `x plus 1`, which would be impossible. Instead we're making an assignment of a new value to the variable `cheap count`. Where the value we're assigning just happens to depend on the previous value of `cheap count`.

Pseudocode

```
pseudocode    for( each car ){
                if(This car's price < $10,000){
                    cheap_count = cheap_count + 1
                }
            }

output  Error: object 'cheap_count' not found
```

Using this variable cheap count, our pseudocode looks like this. But if we tried to write code based on the structure, and put it into R, R would give us an error message saying, object cheap count not found. That's because each time through the loop we're updating the value of cheap count based on its previous value. But the first time through the loop, there is no previous value to base our new value on. So we need to initialize or define the starting value of cheap account before the for loop starts.

Initializing a variable

```
pseudocode cheap_count = ?  
  
for( each car ){  
    if(This car's price < $10,000){  
        cheap_count = cheap_count + 1  
    }  
}
```

So we can do that by writing a line of code that says, cheap count equals some value. Well what value should that be? Remember that cheap count is supposed to tell the number of cheap cars that we've seen so far. And before the for loop starts, we haven't looked at any cars, therefore we can't have seen any cheap cars. So the initial value of cheap count should be 0.

Initializing a variable

```
pseudocode cheap_count = 0

for( each car ){
    if(This car's price < $10,000){
        cheap_count = cheap_count + 1
    }
}

print(cheap_count)
```

The last step of this pseudocode should be to print out the results. We want to know the total number of cheap cars after the for loop is over we've looked at all the cars. So that means we've also seen all of the cheap cars that there are. So the variable cheap count will contain the total number of cheap cars in the data set. So we just need to print out that number.

 SELF-ASSESSMENT FEEDBACK

 CORRECT

What variable would you iterate over for the cars problem?

Your answer: Integers from 1 to the number of cars.

Correct answer: Integers from 1 to the number of cars.

Feedback: Yes! Because we want to work with two variables (Price and Type) inside the for loop, it's best to use a numeric index that refers to which car we're currently examining.

Question 1: What variable would you iterate over for the cars problem?

Correct answer: Integers from 1 to the number of cars.

Feedback: Because we want to work with two variables (Price and Type) inside the for loop, it's best to use a numeric index that refers to which cars we're currently examining.