Statistical Programming with R

Mike Morley // Montclair State University

Purpose

This guide is an informal explanation of what I've learned while using R to analyze a data set for Dr. Hill, as well as a brief overview of the R programming language to help new students get started. These data file that this guide is built around is called "StudentWriteUp.r", and can be located in the "Old Code" folder. This guide is based around the AOC data set only.

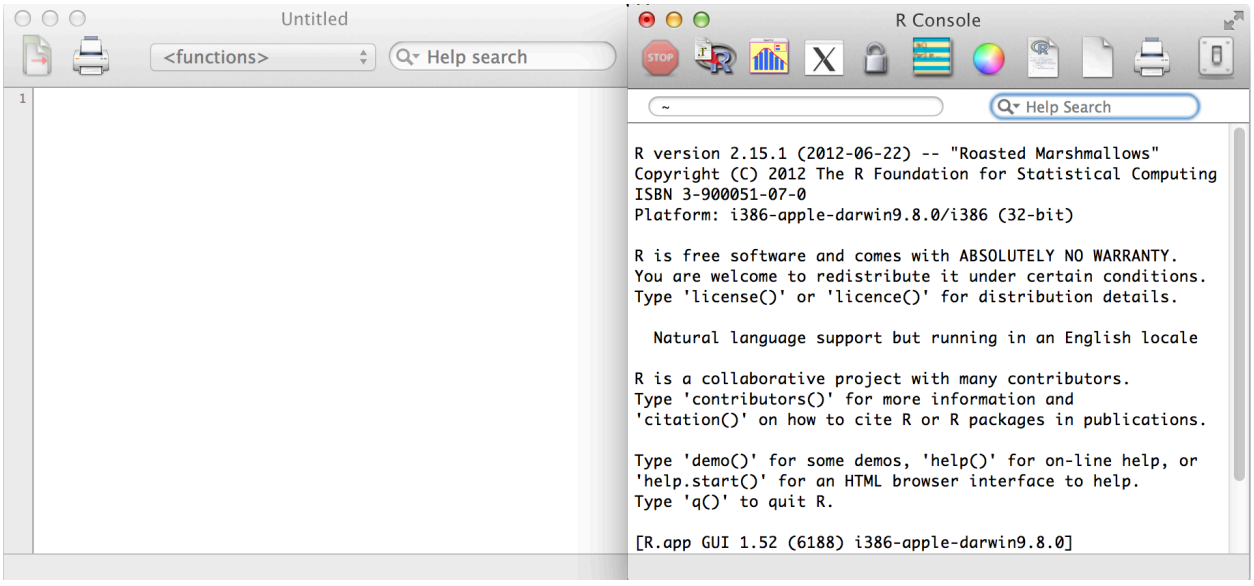
Section 1: Getting Started with R

What is R?

R is an open source statistical programming language. It is freely available at: http://www.r-project.org. It includes binaries for Windows, Mac, and Linux.

Overview of the R console and editor

R will work in two parts: the console, and an editor. On the Mac OS, the built in editor works perfectly fine. This write up is based around using that specific editor. Opening the R icon will open up the R console, where your code/commands can be ran. Using Cmd+N or going to FIle->New Document will open up a new, blank editor. You'll be using the editor to write out your code, and then 'sending' it over to the console where it will be compiled and ran. All output, unless you specify otherwise within your code, will display within the console.



The editor on the left, the R console on the right.

Useful commands/tips to keep in mind:

Cmd+A will select all within the editor.
Cmd+Enter will send all selected code to the console.
Cmd+S will save the current editor file.

Additionally, you can highlight bits of code in the editor and use Cmd+Enter to send only the highlighted code to the console.

You may have noticed you can type commands directory into the R console -- this is fine, if you wish to "code" your analysis one command at a time. It's very useful for "stepping through" and watching what each command will do to your data set. Alternatively, if you wish to write out a program, you'll want to use the editor.

Whatever has been sent to the console is considered the current "workspace". You can save the workspace (this includes all objects/variables created) by using Cmd+S. Alternatively, you can also clear the current R console/workspace by selecting Workspace->Clear Workspace.

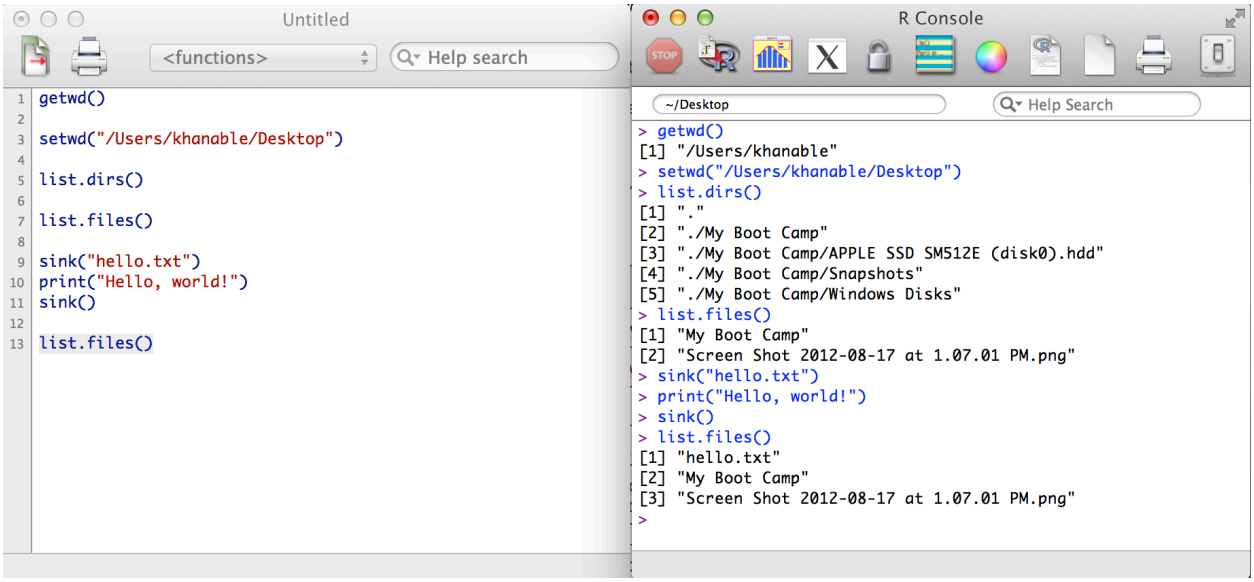
To clear the console of all text, use Cmd+Option+L, or Edit->Clear Console.

Clearing the console will NOT clear current objects/variables.

Basic console commands used in this guide

Action	Command
See current working directory	getwd()
Set current working directory	setwd("PATH")
List files in working directory	list.files()
List directories in working directory	list.dirs()
Create a directory	dir.create("PATH", showWarnings=FALSE) Note: showWarnings=FALSE will stop the console form complaining that the directory doesn't exist.
Assignment operator	<- or = Note: = is bad practice. Doesn't work all the time. Example: x <- 5 (assigns 5 to object 'x')
Bring up help document	?<command> Note: Some commands may need two question marks. Example: ?list.files()
Comments	# prior to text
Print to console	print("TEXT") Note: can also be used with objects, though you can just write the object name into the console to print it.

Action	Command
Output to text file	sink(“PATH.TXT”) Note: All text displayed on the console after sink() will be saved in the specific file. To close writing to that file, add ‘sink()’ (with no path) at the end. Example: sink(“hello.txt”) print(“hello, world!”) sink()
Output to PDF	pdf(“PATH.TXT”) Note: PDF’s are vector images, great for papers. Also, you’ll want to turn off the output device after your text with dev.off() (device off). Example: pdf(“Hello.pdf”) print(“Hello, world!”) dev.off()
Output to PNG	png(“PATH.png”) Note: Use dev.off() at the end
Output to JPG	jpg(“PATH.jpg”) Note: Use dev.off() at the end
Output to BMP	bmp(“PATH.bmp”) Note: Use dev.off() at the end
Get information about an object	str(object) Note: Displays factor (row/col/etc names) and mode information for an object



Example of checking current working directory, setting working directory, viewing directories, viewing files, creating a text file, and then again viewing files.

Section 2: The Data Types

Vectors

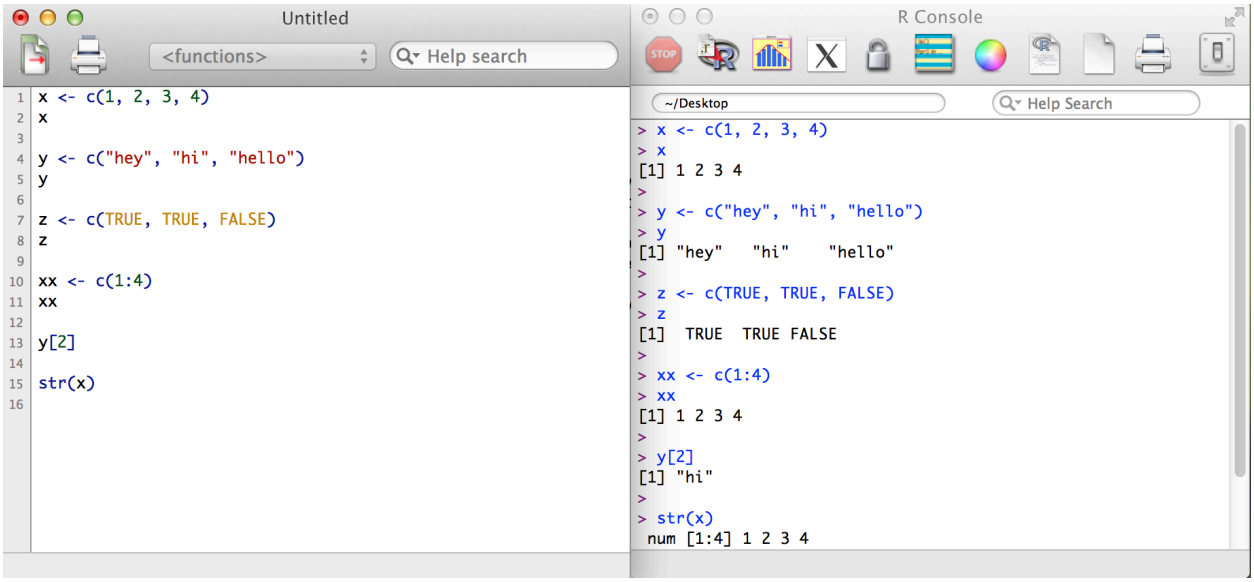
Vectors are one-dimensional arrays that can hold numeric, character, or logical data. The combine function “c()” is used to create a vector.

Examples of a vector:

x<- c(1,2,3,4)
y<-c(“hello”, “hi”, “hey”)
z<-c(TRUE, TRUE, FALSE)

Notes on Vectors

-You cannot mix data types (called “modes”). For example, you can have a vector of character, numeric, or logical data only. You cannot have a vector with both character and numeric data.
-A scalar is simply a vector with a single element.
-Element indices begin at 1, instead of 0 (like most programming languages)
-The colon operator, :, will generate a sequence between numbers. Example x <- c(1:4) is the same as x <- c(1, 2, 3, 4).
-Accessing specific elements is done using brackets; y[2] will return element at index 2, in our case, “hi”.



Example showing the creation of x (numeric vector), y (character vector), and z (logical vector). The object xx illustrates the use of the colon operator, and str(x) gives information about the object.

For more information about vectors, type ?vector into the R console.

Matrices

A matrix is a two dimensional (rows and columns) array where each element is of the same mode (numeric, character, or logical).

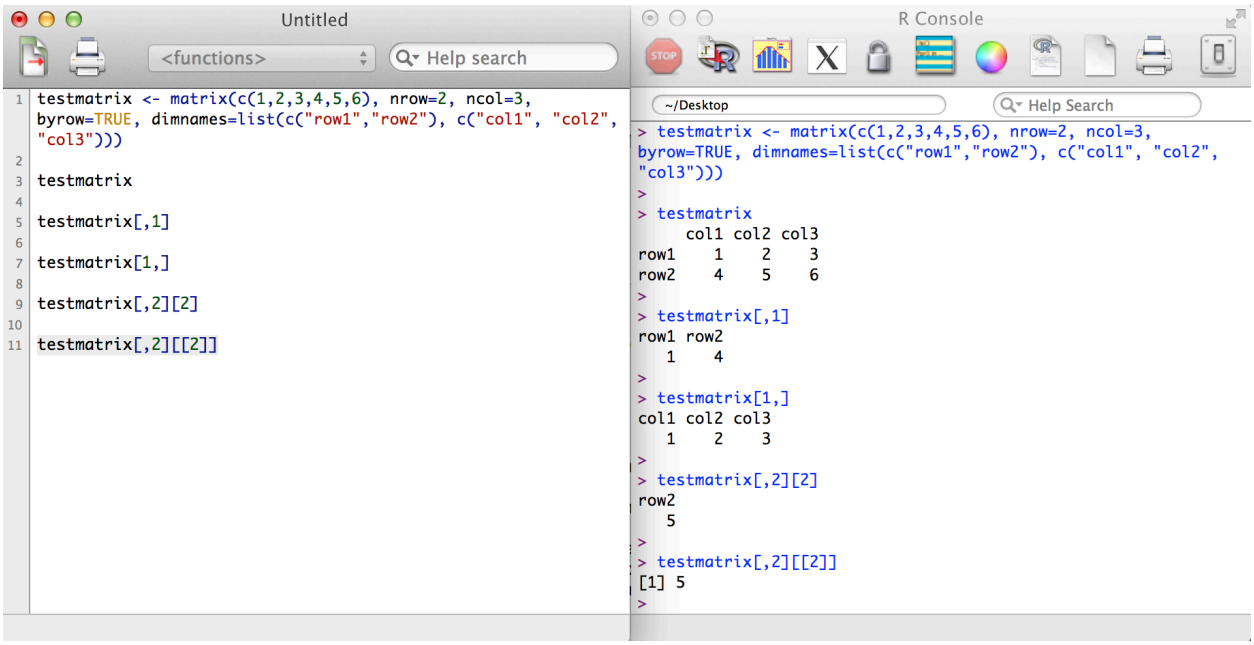
Example of a matrix:

testmatrix <- matrix(DATA, nrow=#ROWS, ncol=#COLS, byrow=TRUE/FALSE, dimnames=list(c("ROWNAMES"), c("COLNAMES")))

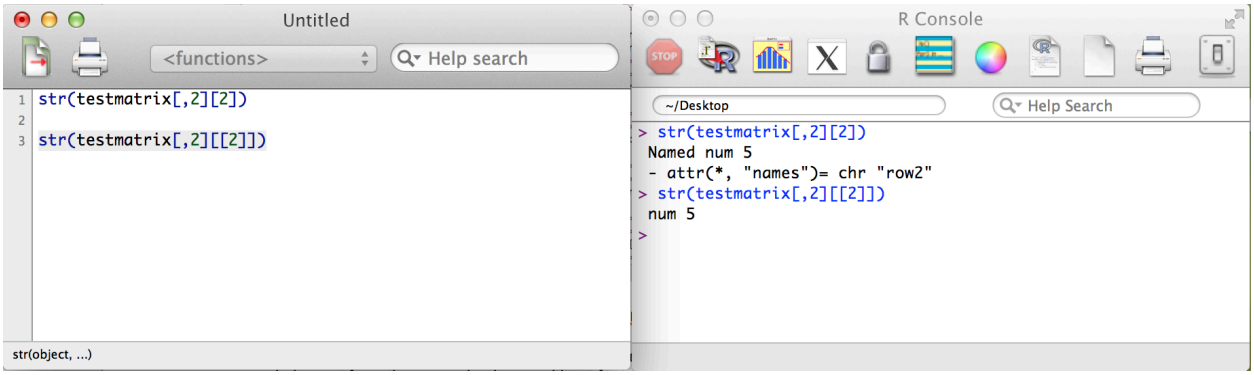
Where DATA will be the data, nrow will be the amount of rows, ncol will be the amount of columns, byrow can be true or false (populate the matrix by row if true, or column if false), and dimnames will be the names of the rows and columns.

Notes on Matrices

-Recalling a specific element in a matrix will now require the use of a comma. For example, testmatrix[,1] will recall the entire first column, and testmatrix[1,] will recall the entire first row. To recall only one element, you'll need to use two sets of brackets. Example: testmatrix[,2][2] will recall the second element in the second column.
-Using single brackets will recall elements and retain factor information (ie, row/column names). To return just the element with no factors attached, use two brackets. Example: testmatrix[,2][[2]] will return the second element of the second column, without factor information. This will be useful when you want to return just a scalar.
-A matrix must contain data of the same mode (character, numeric, logical)



Example of creating a 2x3 matrix. Note how using testmatrix[,1] recalls the entire first column, and testmatrix[1,] recalls the entire first row. Making use of two sets of brackets, testmatrix[,2][2] recalls the second element from the second column. Using double brackets, testmatrix[,2][[2]] recalls only the second element from the second column, without factor information ("row2" is now missing from output).



Example of how using double brackets will return a scalar with no factor information.

For more information about matrices, type ?matrix into the R console.

Arrays

Arrays are very much like vectors or matrices, but they posses the ability to have one, two, or more dimensions. A two dimensional array is a matrix, but a one dimensional array isn't a vector. Different functions may require a one dimensional array as an argument as opposed to a vector.

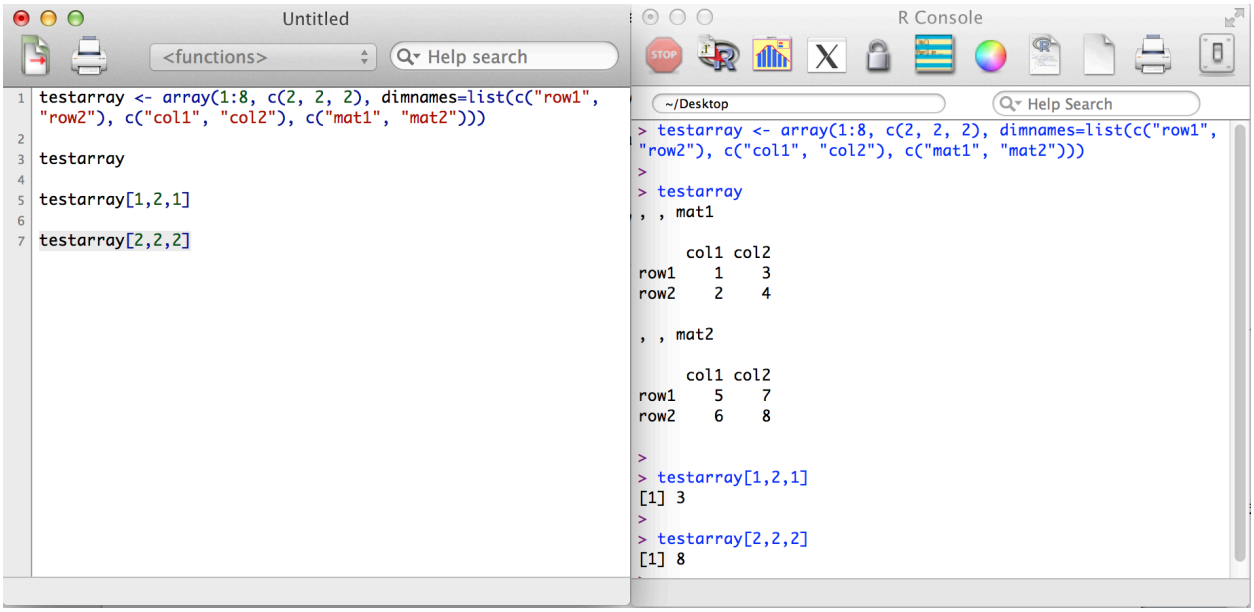
Example of an Array:

testarray <- array(DATA, DIMS, dimnames=list(c("ROWNAMES), c("COLNAMES"), c("OTHERNAMES")))

Where DATA will be your data, DIMS will be the dimensions of the array (given in c() form), and dimnames will be a list of the dimensions names.

Notes on Arrays

-Recalling specific elements is in the form of: arrayname[row_#, element_in_that_row, mat_#]. For example: testarray[1,2,1] will recall the second element in the first row, of the first matrix.



Example of creating an array, and using the element notation to locate specific elements.

For more information about arrays, type ?array into the R console.

Data Frames

A data frame is much like a two dimensional matrix, except it has the ability to contain more than one mode of data. For example, one column could have character data, while the next column contains numeric data. Data frames are the backbone of working in R.

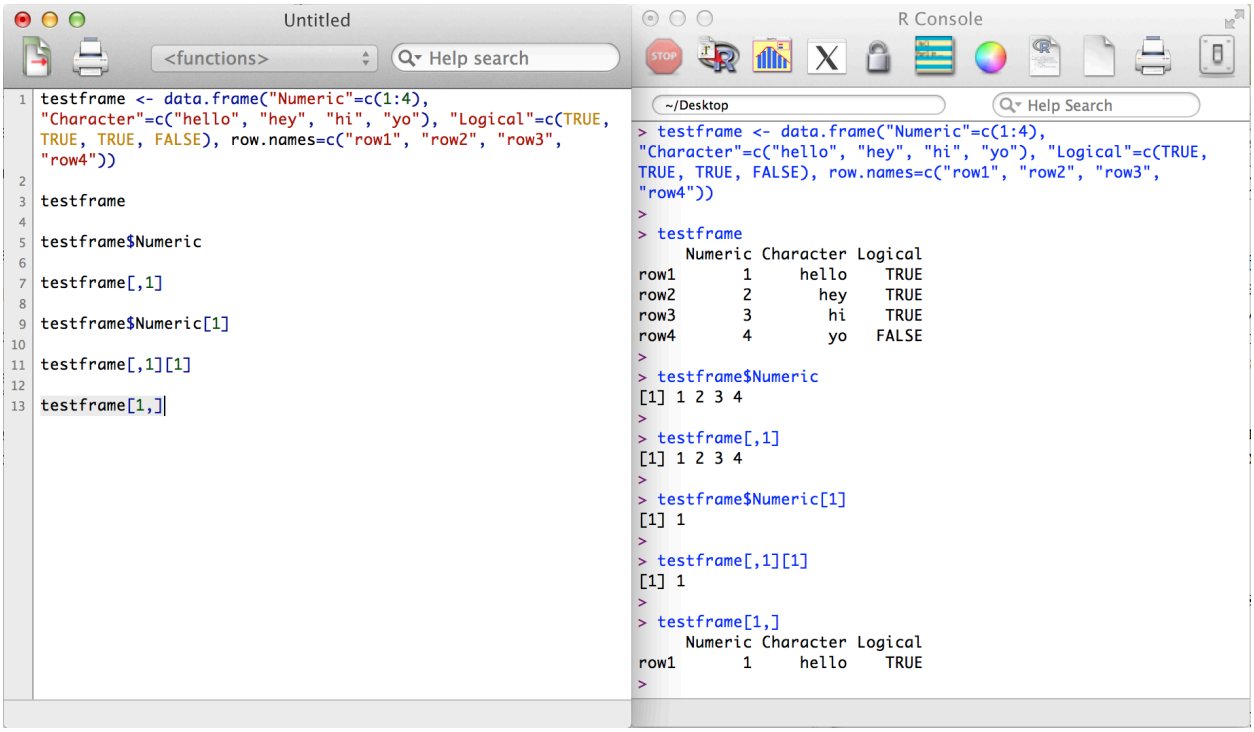
Example of a Data Frame:

testframe <- data.frame("COLNAME"=c(COLDATA), "COLNAME2"=c(COLDATA2), row.names=c("ROWNAMES"))

Where COLNAME will be replaced by the name you wish to give to each column, COLDATA will be the data for that specific column, and the ROWNAMES will be a list of names for the rows.

Notes on Data Frames:

-Each column can only contain one type of data (numeric, character, logical)
-Recalling specific columns in a data frame can be handled using the dollar sign operator, $. For example, testframe$Numeric will return the column named "Numeric" within testframe. We can also use our numeric indices as well. For example, testframe[,1] will return the first column of testframe.
-Recalling specific elements in a data frame can be handled using both numeric indices and the dollar sign operator. For example, testframe$Numeric[1] returns the first element in the column named "Numeric" within testframe. This is equivalent to using testframe[,1][1], the first element of the first column of testframe.



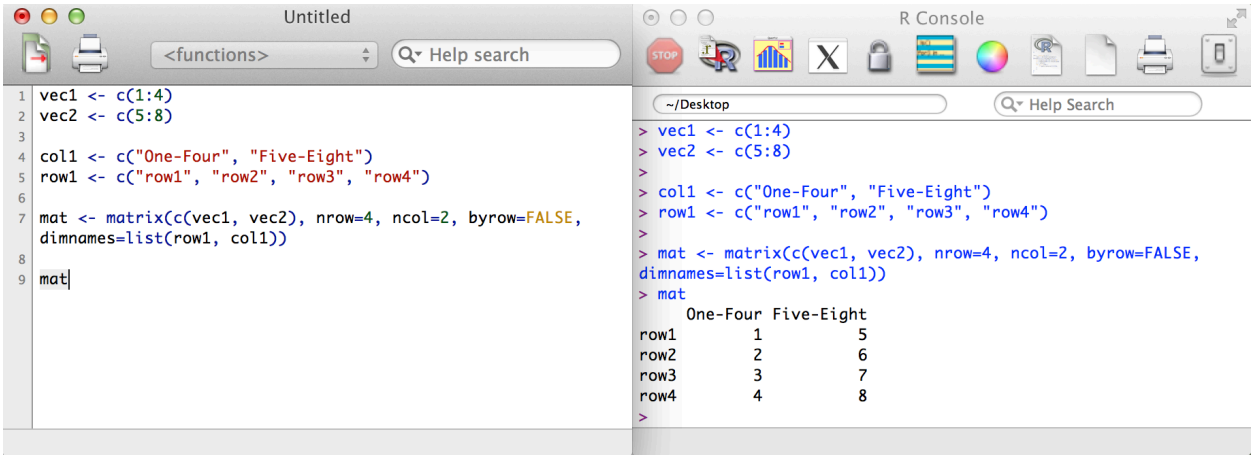
An example showing the creation of a data frame, testframe, and the creation of Numeric, Character, and Logical columns as well as row1-4. Also shows an example of using the dollar sign operator, $, to display specific columns.

Playing with data types

Constructing objects (vectors, matrices, arrays, data frames) doesn't need to be done all at once like I've done with all of the examples. We can, in fact, create them piece by piece from other objects. Here are some examples:

<u>Creating a matrix from vectors</u>

We are able to add vectors into a matrix very easily. We simply create our desired vectors, and then use their object names when creating the matrix:



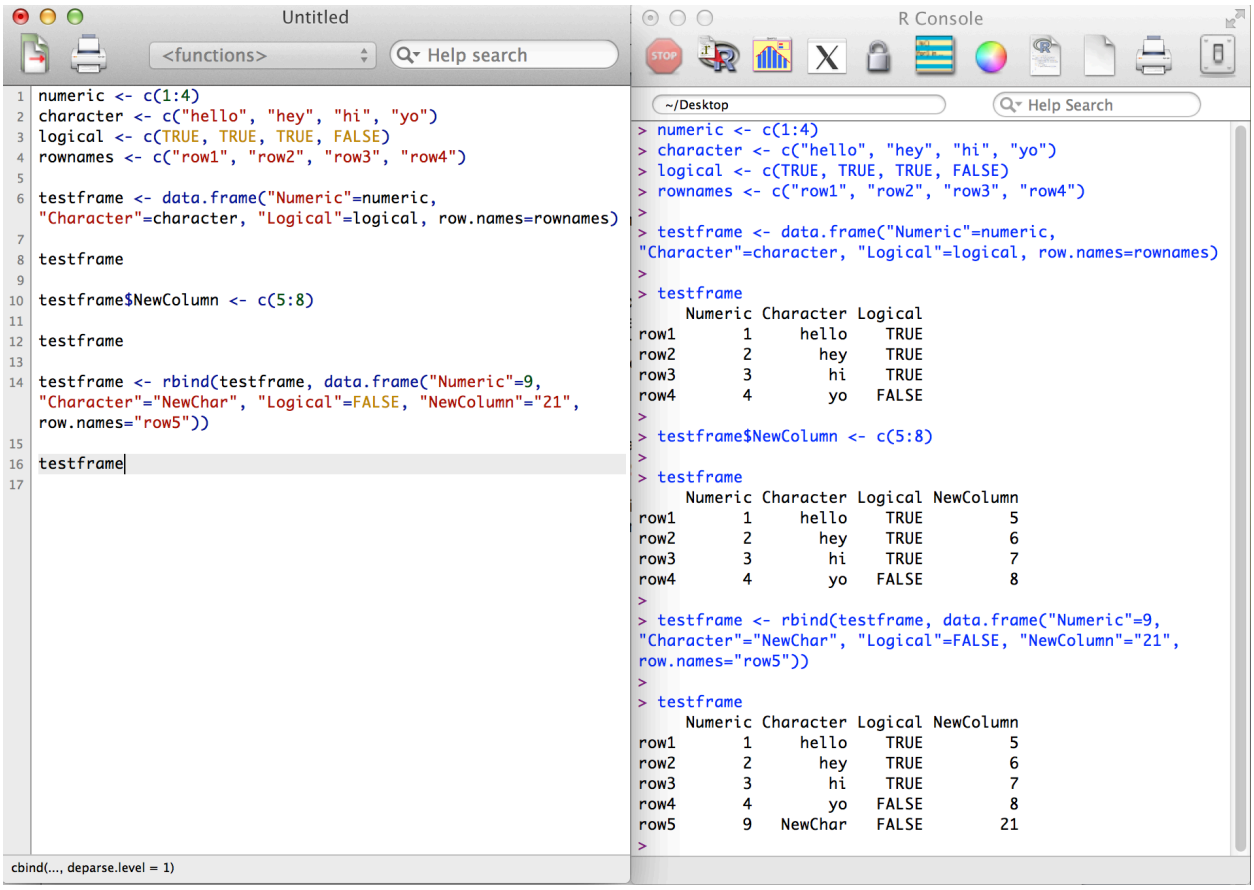
An example of populating a matrix using vectors of data, row names, and column names.

Adding vectors to a data frame

We can also add vectors into a data frame by using their object name much like we did with a matrix. Some other interesting things that we can do with a data frame is the creation of a new column or row, specifically by using the dollar sign operator, $, for a new column (by assigning the new column with a vector of data), or by using the rbind() function for creating a new row.

rbind()

rbind() is a function within R that allows you to join two data frames by row, which is how we will go about adding rows to an existing data frame. The function works as follows: dataframename <- rbind(dataframename, data.frame(new data frame arguments)).

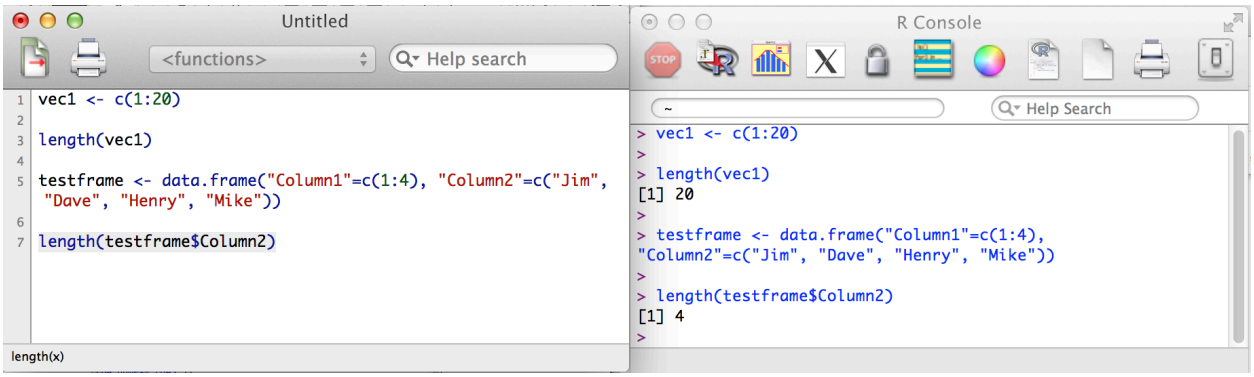


An example of populating a data frame by using vectors. Also illustrates how the dollar sign operator can be used to create a new column, and how rbind() adds a row into the data frame by joining a second data frame to the bottom.

There also exists another function, cbind(), which allows for the joining of two data frames by columns.

Checking the length of a vector, column, or row

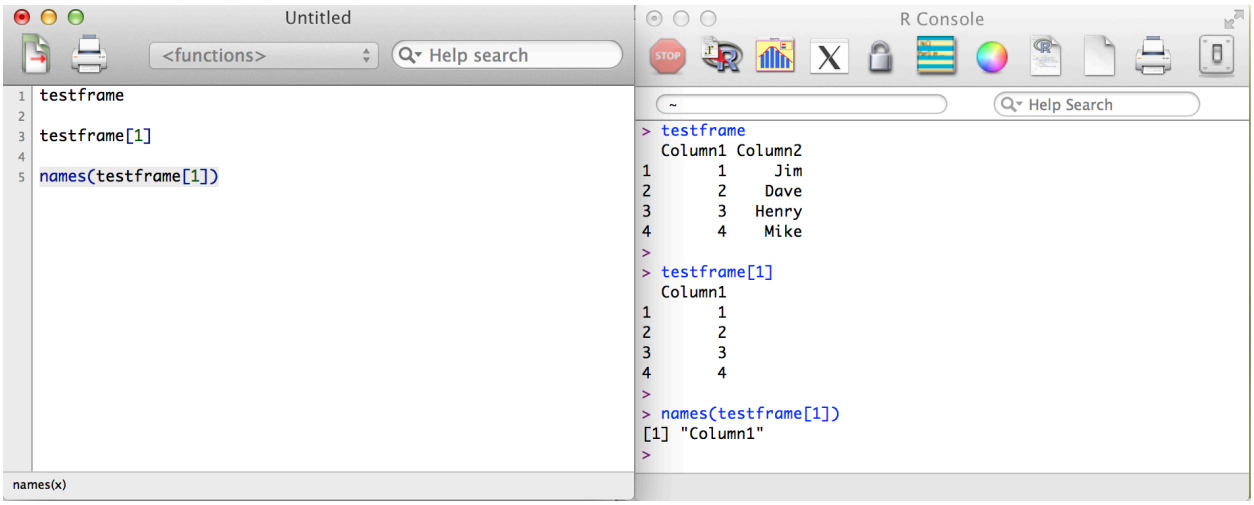
To get the length of a vector, column, or row (very useful in loops) we simply use the length() function.



An example showing how to check the length of a vector and column.

Getting the name of a specific column or row

Getting the name of a column or row will be done by using the names() function.



Reordering data in a Data Frame

Reordering a data frame can be done using the reorder() function along with the with() function.

The general function looks like this:

testframe <- with(testframe, reorder(COLUMN_TO_BE_ORDERED, COLUMN_TO_BE_ORDERED_BY, ORDERING_FUNCTION)

Where COLUMN/ROW_TO_BE_ORDERED will be the column that will receive the ordering, COLUMN_TO_BE_ORDERED_BY will be the column that the ordering function will order by, and the ordering function is a function (such as mean, median, etc -- see section 4 for statistical and mathematical functions) used to order.

An example of this can be found in my overview of the analysis.

Section 3: Loops, ifs, and logical statements

If statements

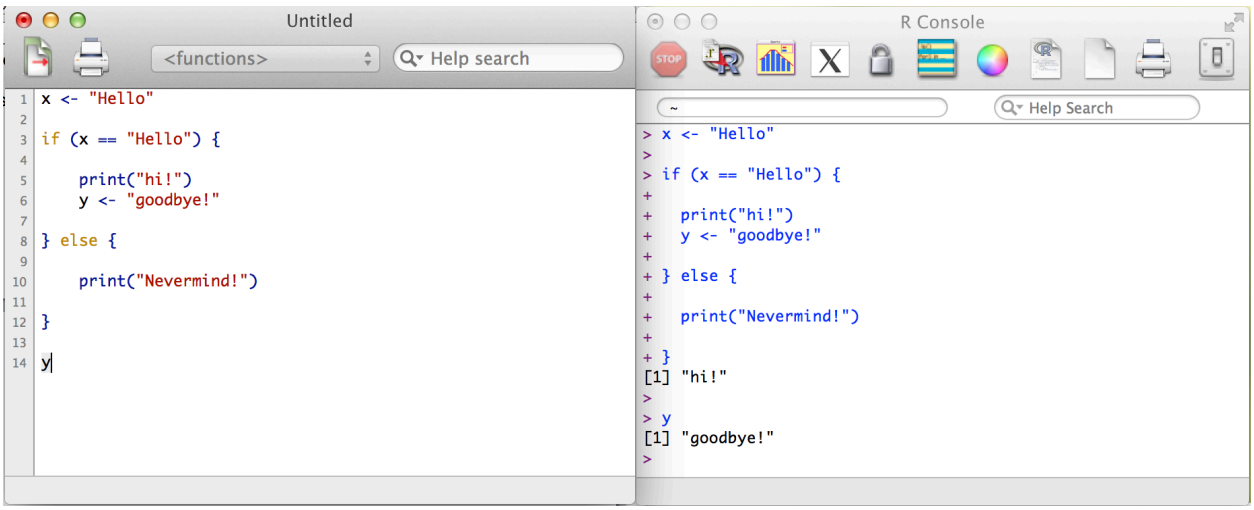
If statements in R work very much like if statements in Java. They're of the form:

if (condition) dothis else dothis

We can write out the entire statement on one line if we wish. For example:

x <- "Hello"
if (x == "Hello) print("hi!") else print("goodbye!")

In order to make use of more than one statement we will need to use curly brackets:



All statements between the curly brackets will be treated as one single statement.

Logical Operators

Operator	Logical Statement
Equal to	==
Not equal to	!=
Less than or equal	<=
Less than	<
Greater than or equal	>=
Greater than	>
And	&&
Not	!
Or	||

The logical operators are the same as the ones used in Java.

For/While Loops

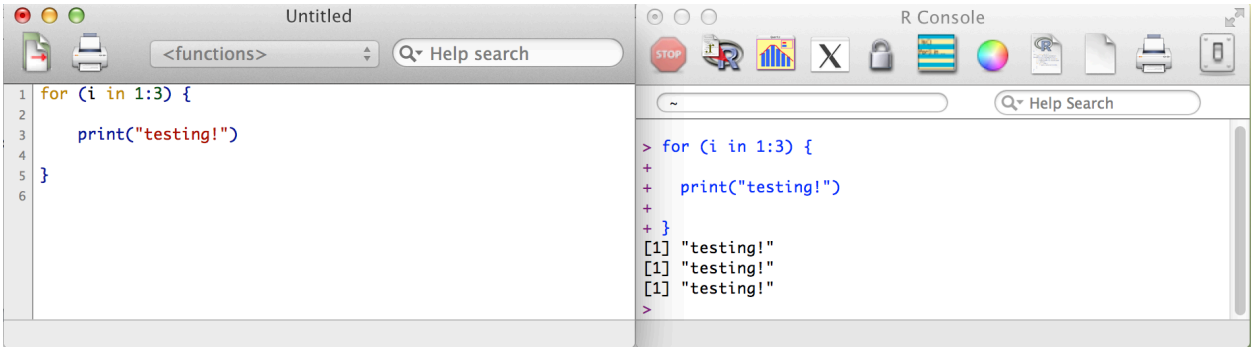
The for loop in R follows this basic setup:

for (i in 1:4) {
statements
}

Where “i” can be any variable, and the colon operator is read as “to”. Thus we have for some variable in 1 to 4, do these statements. Easy!
While loops work as expected and follow this template:

while (condition is true) statements

Where additional statements can be entered by using curly brackets.

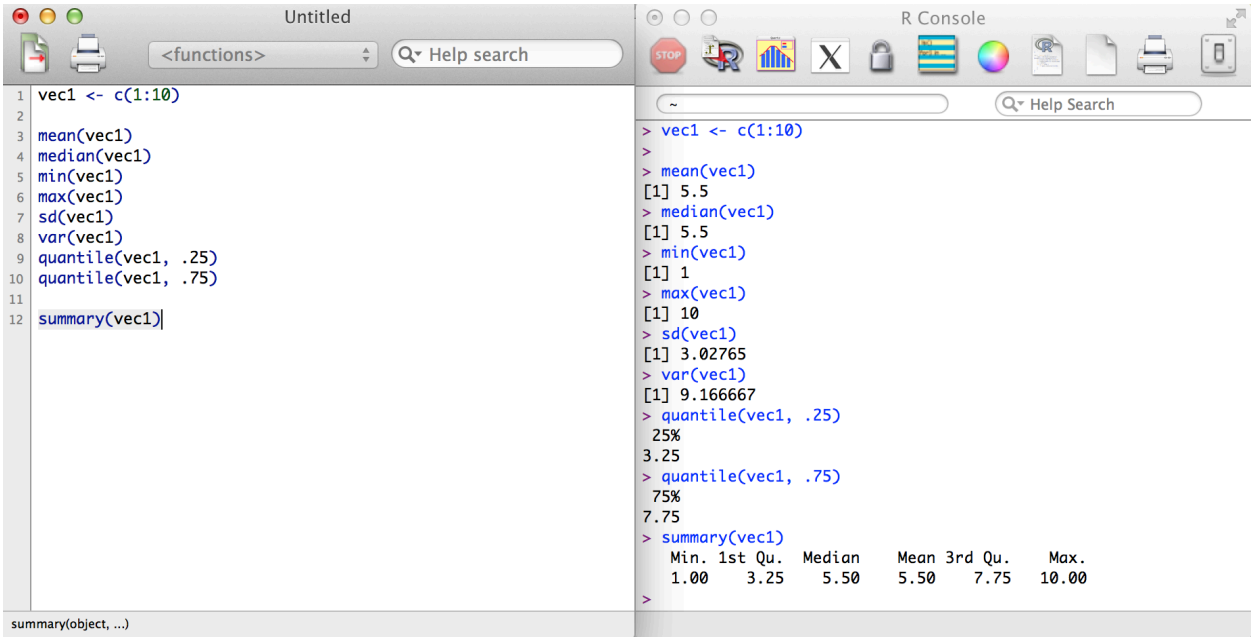


An example of a for loop.

Section 4: Statistical functions

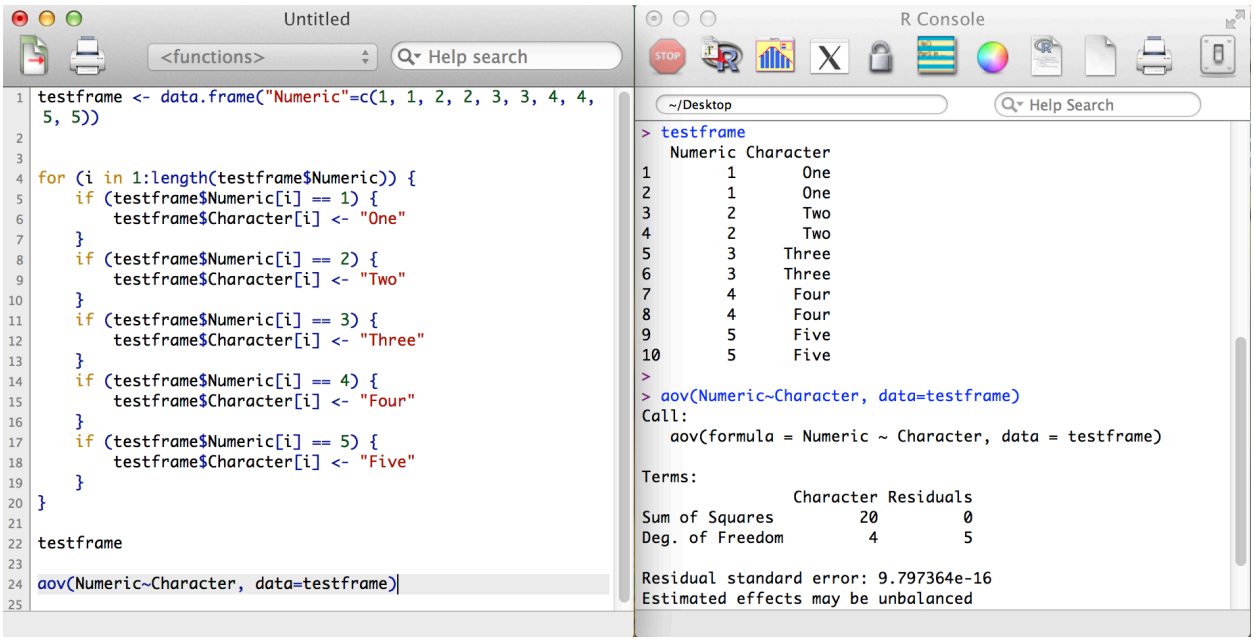
Function	Details
summary()	Returns summary statistics for an object
mean()	Returns the mean of an object
median()	Returns the median of an object
min()	Returns the minimum value in an object
max()	Returns the maximum value in an object
var()	Returns the variance of an object
sd()	Returns the standard deviation of an object
quantile(x, .75)	Returns the third quartile of an object

Function	Details
quantile(x, .25)	Returns the first quartile of an object
quantile(x, .50)	Returns the second quartile of an object (this is the same as the median)
aov()	Analysis of variance
TukeyHSD()	Tukey HSD test (requires an aov object)
boxplot()	Creation of a boxplot
plot()	General plot command



An example of the statistical methods.

The aov() function requires the use of a data frame, and is in the form of aov(formula=COLNAME~COLNAME, data=DATAFRAME), the boxplot() function is in the form of boxplot(formula=COLNAME~COLNAME, data=DATAFRAME). TukeyHSD() will simply take a aov object, for example if we ran an aov test and saved it into an object “x”, the command TukeyHSD(x) will run the Tukey HSD test on the aov object “x”.



An example of creating a data frame, testframe, with a column named Numeric, and then using a for loop and if statements to assign the Character column by what exists in the Numeric column. Next, we use the aov function to analyze the data contained within the data frame.

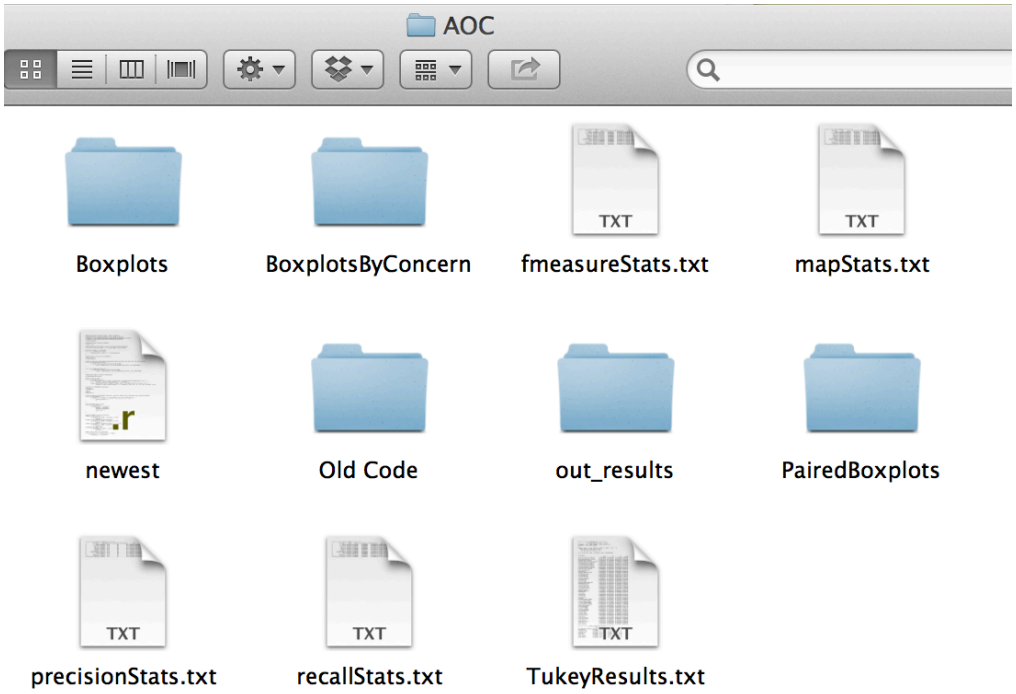
TukeyHSD() can be passed an aov object for analysis, so simply saving our aov into an object (for example, x <- aov(...)) and then passing x to TukeyHSD(x) will return the TukeyHSD results.

Section 5: Overview of my code

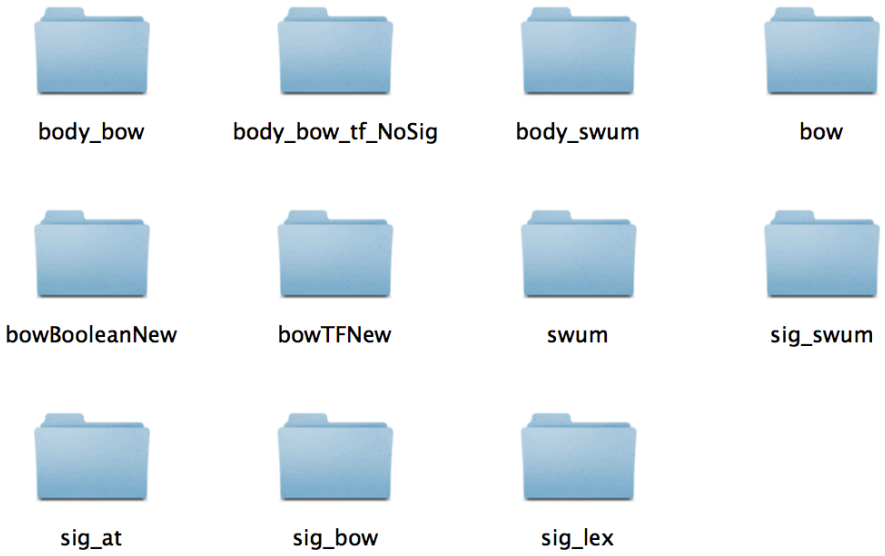
My independent study with Dr. Hill over the summer of 2012 had two core goals: for me to learn R, and for me to apply my new skill set to the analysis of Dr. Hills research.

Layout of the data

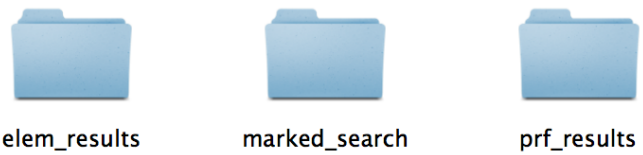
The data I analyzed was in the form of many .out files, which contained text information on concerns for a specific technique. There were three folders of .out files for each technique -- a marked_search folder, which contained a list of the returned queries as well as their ‘score’, and an asterisk next to the returned queries which were relevant to the search, a prf_results folder, which contained precision, recall, F-measure, and related results for each query. There was a third folder, elem_results, but I did not make use of the files contained within this folder.



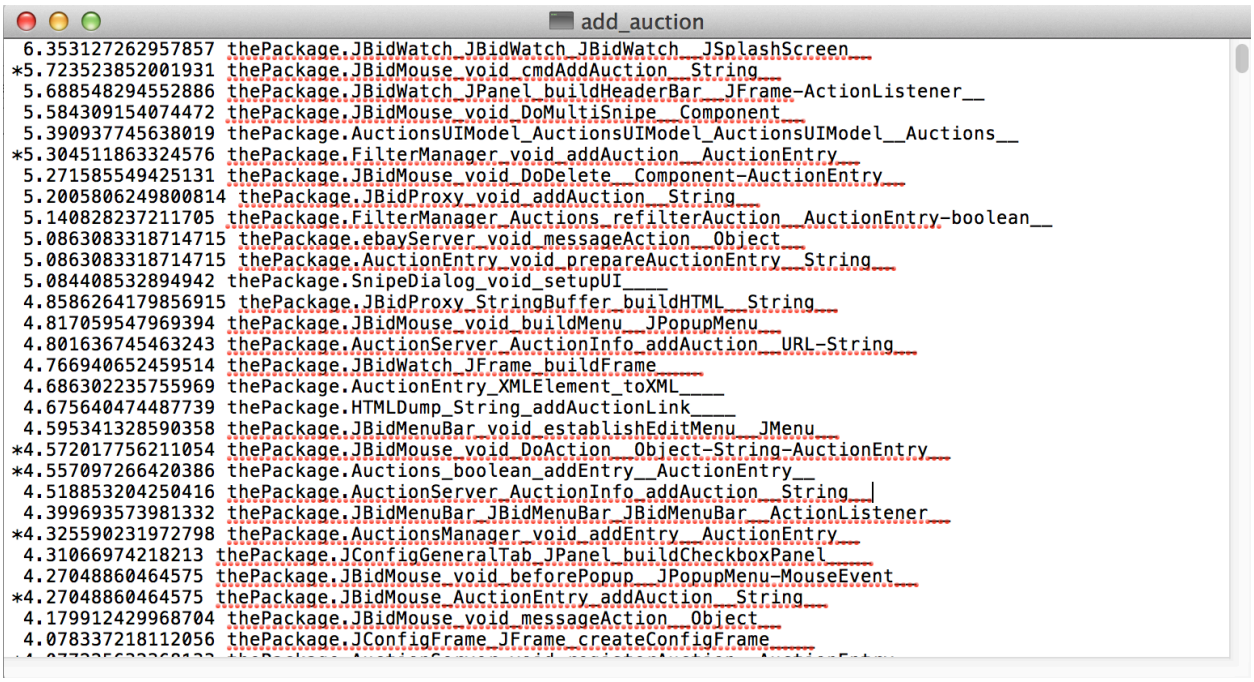
My workspace, the AOC folder. Within it contains the out_results folder, which stored folders for each technique. The remaining files are either my R code, or output from my R code. More on this later.



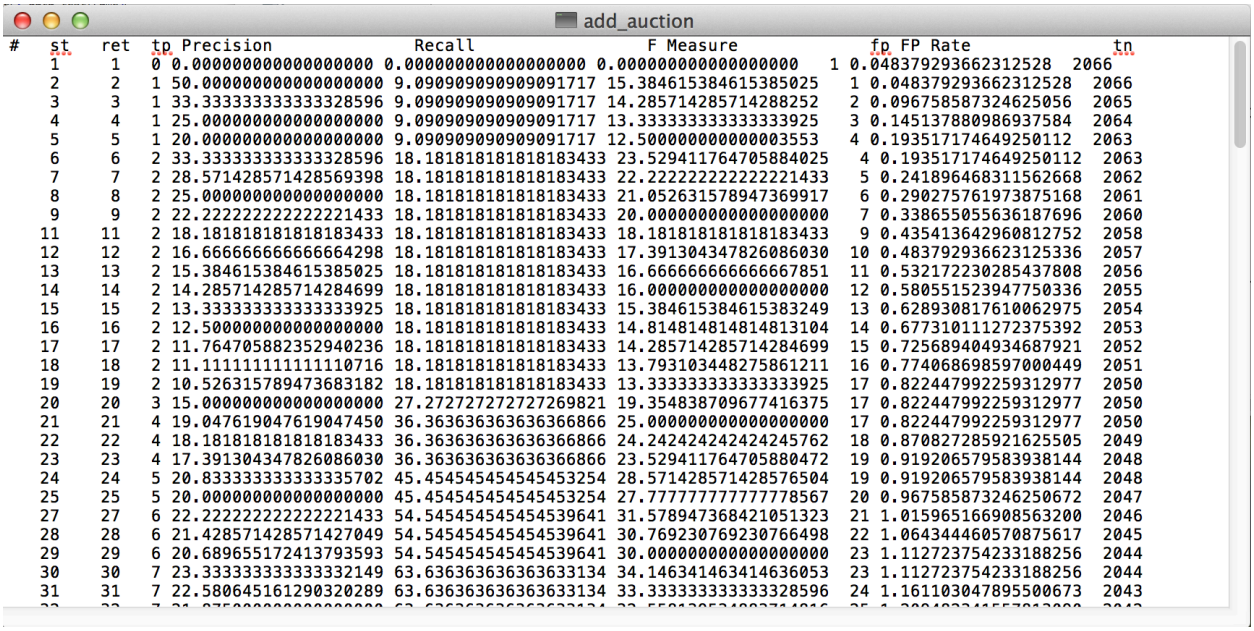
Folders for each technique within the out_results folder.



Marked_search, prf_results, and elem_results (these three folders are included in each of the technique folders above).



An example of a .out file from the marked_search folder - add_auction.out, which contained the returned queries and the scoring result for this specific concern/technique combination. Note the asterisks - these are relevant results.



#	st	ret	tp	Precision	Recall	F Measure	fp	FP Rate	tn
1	1	0	0.0	0.0000000000000000	0.0000000000000000	0.0000000000000000	1	0.048379293662312528	2066
2	2	1	50.0	0.000000000000000000	9.090909090909091717	15.384615384615385025	1	0.048379293662312528	2066
3	3	1	33.333333333333328596	9.090909090909091717	14.285714285714288252	2	0.096758587324625056	2065
4	4	1	25.0000000000000000	9.090909090909091717	13.333333333333333925	3	0.145137880986937584	2064
5	5	1	20.000000000000000000	9.090909090909091717	12.500000000000003553	4	0.193517174649250112	2063
6	6	2	33.333333333333328596	18.181818181818183433	23.529411764705884025	4	0.193517174649250112	2063
7	7	2	28.571428571428569398	18.181818181818183433	22.222222222222221433	5	0.241896468311562668	2062
8	8	2	25.000000000000000000	18.181818181818183433	21.052631578947369917	6	0.290275761973875168	2061
9	9	2	22.222222222222221433	18.181818181818183433	20.000000000000000000	7	0.338655055636187696	2060
11	11	2	18.181818181818183433	18.181818181818183433	18.181818181818183433	9	0.435413642960812752	2058
12	12	2	16.666666666666664298	18.181818181818183433	17.391304347826086030	10	0.483792936623125336	2057
13	13	2	15.384615384615385025	18.181818181818183433	16.666666666666667851	11	0.532172230285437808	2056
14	14	2	14.285714285714284699	18.181818181818183433	16.000000000000000000	12	0.580551523947750336	2055
15	15	2	13.333333333333333925	18.181818181818183433	15.384615384615383249	13	0.628930817610062975	2054
16	16	2	12.500000000000000000	18.181818181818183433	14.814814814814813104	14	0.677310111272375392	2053
17	17	2	11.764705882352940236	18.181818181818183433	14.285714285714284699	15	0.725689404934687921	2052
18	18	2	11.111111111111110716	18.181818181818183433	13.793103448275861211	16	0.774068698597000449	2051
19	19	2	10.526315789473683182	18.181818181818183433	13.333333333333333925	17	0.822447992259312977	2050
20	20	3	15.000000000000000000	27.272727272727269821	19.354838709677416375	17	0.822447992259312977	2050
21	21	4	19.047619047619047450	36.363636363636366866	25.000000000000000000	17	0.822447992259312977	2050
22	22	4	18.181818181818183433	36.363636363636366866	24.242424242424245762	18	0.870827285921625505	2049
23	23	4	17.391304347826086030	36.363636363636366866	23.529411764705880472	19	0.919206579583938144	2048
24	24	5	20.833333333333335702	45.454545454545453254	28.571428571428576504	19	0.919206579583938144	2048
25	25	5	20.000000000000000000	45.454545454545453254	27.777777777777778567	20	0.967585873246250672	2047
27	27	6	22.222222222222221433	54.545454545454539641	31.578947368421051323	21	1.015965166908563200	2046
28	28	6	21.428571428571427049	54.545454545454539641	30.769230769230766498	22	1.064344460570875617	2045
29	29	6	20.689655172413793593	54.545454545454539641	30.000000000000000000	23	1.112723754233188256	2044
30	30	7	23.333333333333332149	63.636363636363633134	34.146341463414636053	23	1.112723754233188256	2044
31	31	7	22.580645161290320289	63.636363636363633134	33.333333333333328596	24	1.161103047895500673	2043

An example .out file from the prf_results folder, add_auction.out. This contained rank, true positive, precision, recall, f-measure, false positive, and true negative information.

The goal from this point was to read in the information from all of these files and store them in a single data frame complete with precision, recall, f-measure, and MAP (mean average precision) results for each technique/concern/query combination.

In order to calculate precision and recall correctly, we needed to have a pre-defined threshold value - a sort of 'cut off' value. We determined 10 would be best, so precision and recall would be calculated using the first 10 results for each .out file within each marked_search folder.

To start, I defined scalars that held the path to where the out_results folder was within my working directory, as well the path to where all the output would be placed.

```
#Working directory (contains data), change to whatever
#Output directory (outputs boxplots, etc), change to whatever
workingdir<-"/Users/khanable/Documents/CMPT495/workspace/AOC/out_results"
outputdir<-"/Users/khanable/Documents/CMPT495/workspace/AOC"
setwd(workingdir)

#threshhold value, change to whatever
thresh <- 10
```

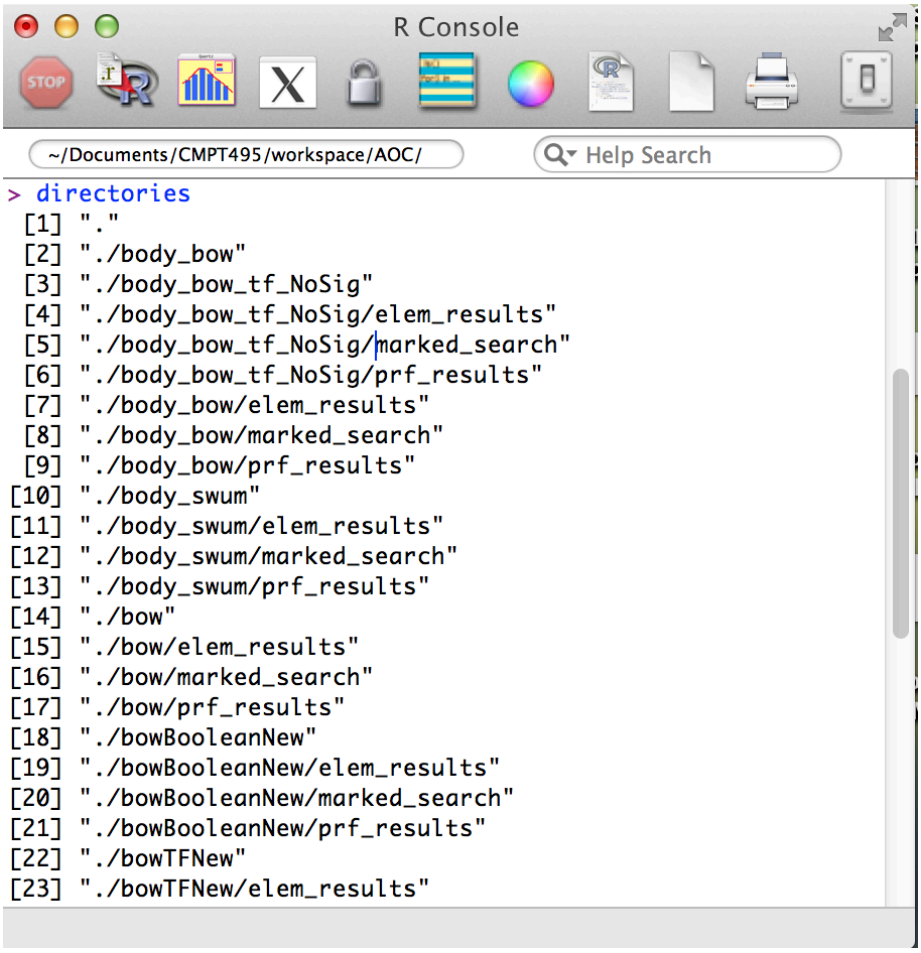
The thresh scalar contains the number of our threshold, in this case, 10. Remember, we're calculating precision and recall based on the first 10 entires.

Now I needed to read in the data from all the files within the marked_search folder, in order to accomplish this I created a vector named directories that would contain the path information for each directory within out_results:

```
#scans directory and creates a list of all files and sub directories
directories<-list.dirs(path=".", full.names=TRUE, recursive=TRUE)
```

You'll notice the arguments full.names=TRUE (which will force the inclusion of the entire path name), and recursive=TRUE, which will not only included the directories contained in out_results, but all of their subdirectories as well.

This is a sample of the output from the directories object:

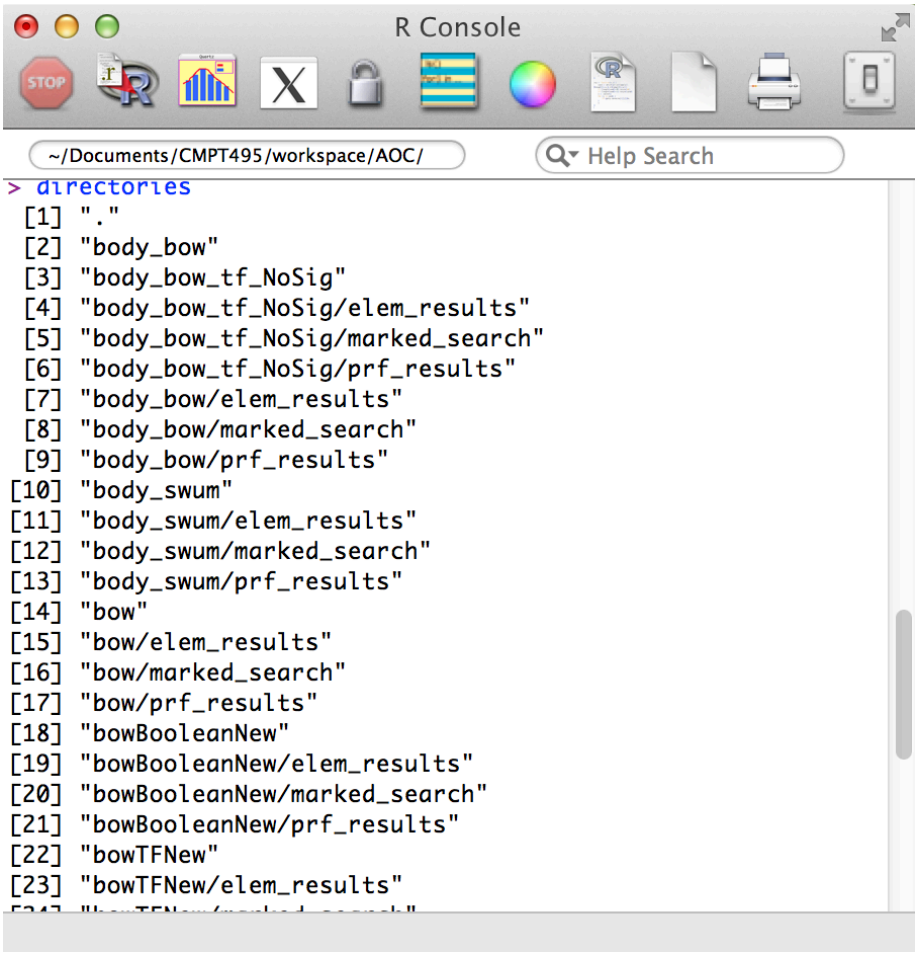


Next I needed to rid each element of the directories vector of the "./" that precedes each entry. I used a for loop and the sub() function to substitute "./" with nothing ("").

```
#a loop to remove "./" from path
for (i in 1:length(directories)) {
 directories[i] <- sub("./", "", directories[i])
}
```

Note how I use the length() function to determine the length of my loop, and brackets on the end of directories[i] to specify each element (which will be determined by the loop).

This is a sample of what the directories object now contained after the substitution:



Much easier to work with now that "./" has been removed.

Now that I had a complete list of the directories and subdirectories, I could work on getting a list of all the individual files stored within all of these directories. I started with two empty vectors, files and files2:

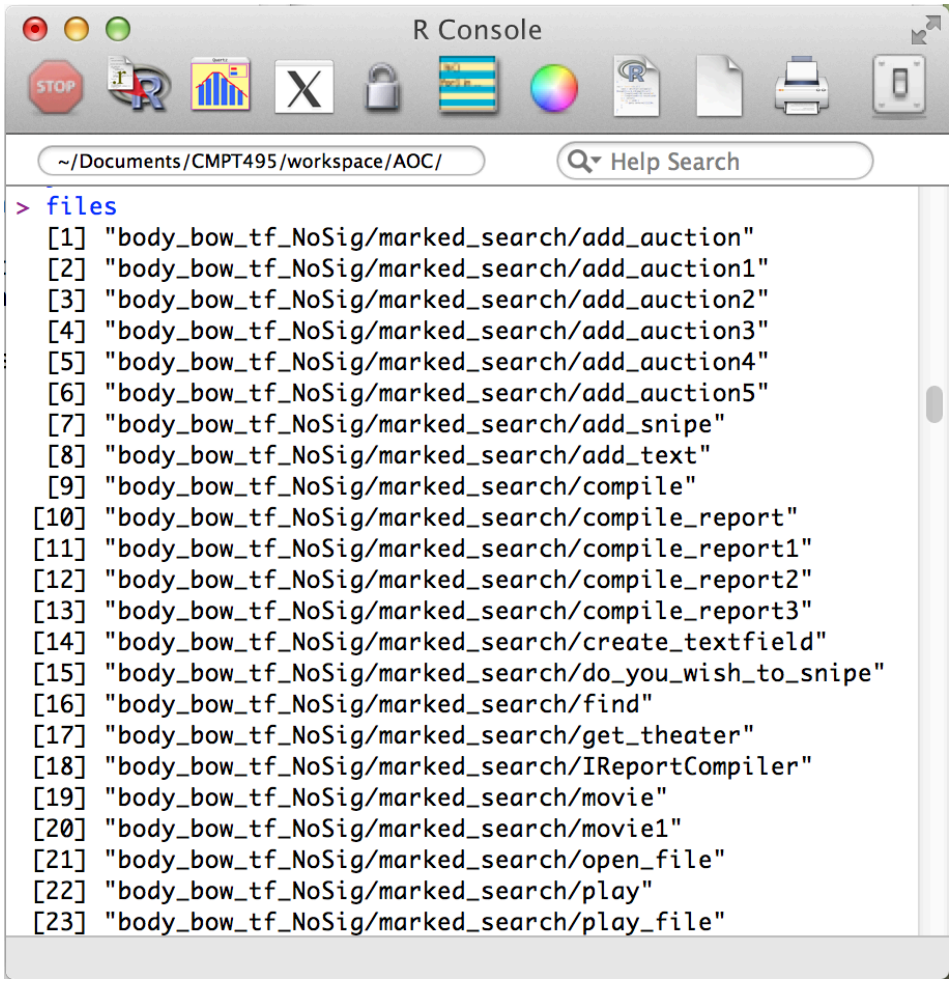
```
#empty vectors to store file information
files<-vector()
files2<-vector()

#loops to store all files within marked_search and prf_results into the files and
 files2 vectors
for (i in 1:length(directories)) {
 if (grepl("marked_search", directories[i])==TRUE)
      files <- append(files, list.files(path=directories[i], full.names=TRUE))
}

for (i in 1:length(directories)) {
 if (grepl("prf_results", directories[i])==TRUE)
      files2 <- append(files2, list.files(path=directories[i], full.names=TRUE))
}
```

Each of these loops is cycling through the entirety of our directories vector, using the grepl() function (a function that will look for a specific pattern of characters within a string, and return true it if exists) to

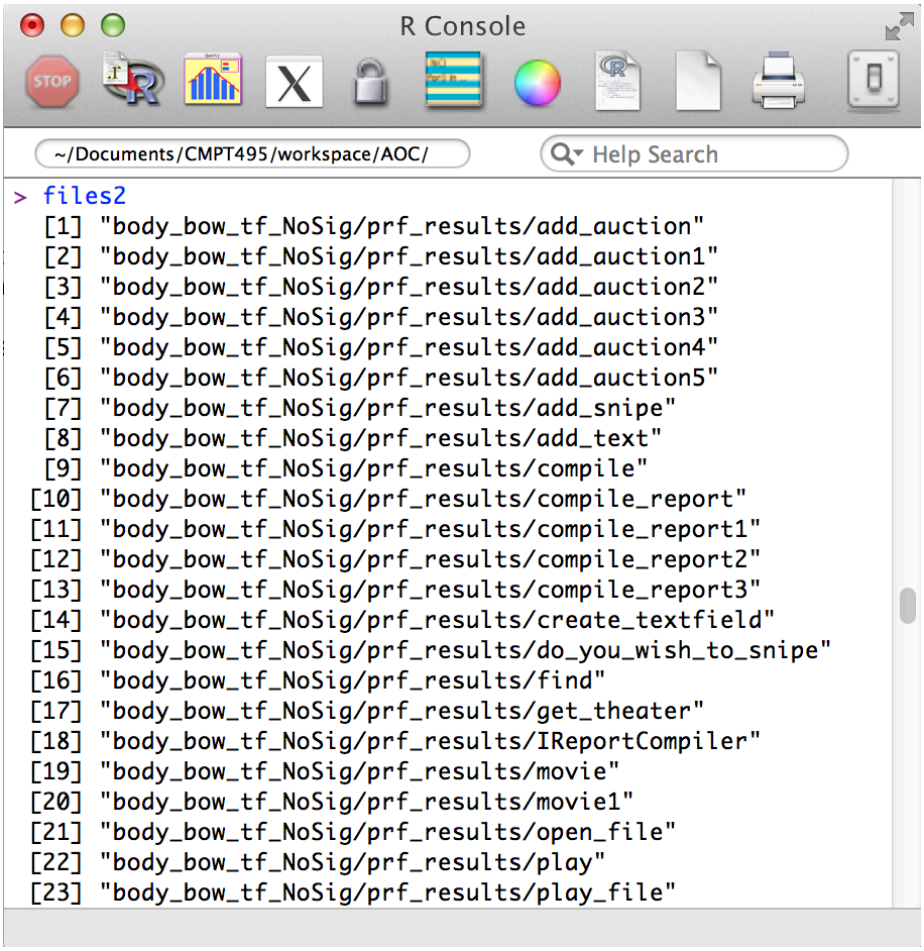
locate directories with “marked_search” and “prf_results” in the path. If the path returns true (in that the path contains the string “marked_search” or “prf_results”), our files and files2 vectors will then use the list.files() function to store the names of each of the files within the specific directories. Having the argument full.names=TRUE will allow us to have the full path to each file. The append() function was used to append new data to the end of the files or files2 vectors, it works as follows:

append(VECTOR, WHAT_TO_APPEND)

Here is what our files and files2 vectors looked like:



The files vector, containing the full path to every file within the marked_search folders for each technique.



The files2 vector, containing the full path to every file within the prf_results folders for each technique.

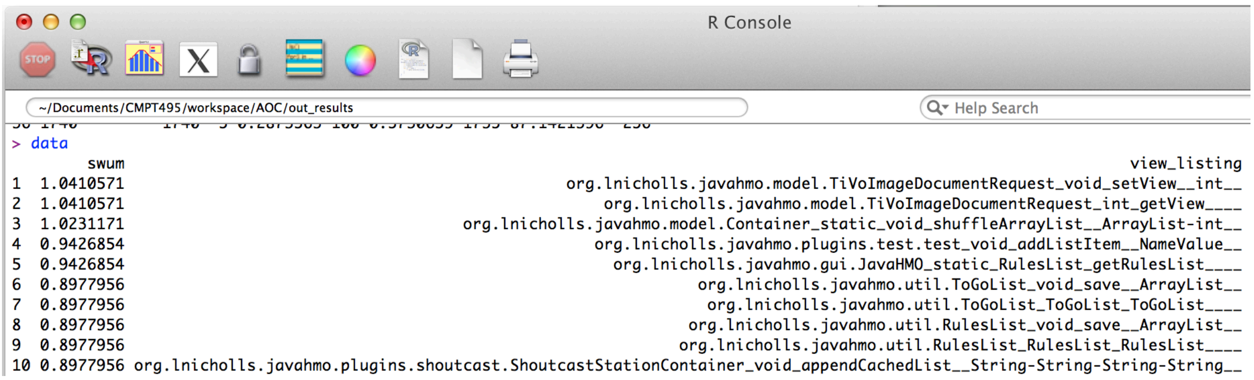
So now we have two vectors that contain lists of all the files that we will need to read in. We're going to need to read in the information contained in each of the files listed, so we're going to use a data frame as the .out files contain both numeric and character data.

```
prfdataframe<-data.frame()

#loop to read in files
for (i in 1:length(files)) {
 data<-read.table(files[i], sep="", nrows=thresh, col.names=c(sub(".marked_search.
 (.*)", "", files[i]), sub("(.*?)marked_search.", "", files[i])))
 data2<-read.table(files2[i], header=F, skip=1, sep="", col.names=c(sub(".prf_results.
 (.*)", "", files2[i]), sub("(.*?)prf_results.", "", files2[i]), "tp", "p", "r", "f",
 "fp", "fp-rate", "tn"))
```

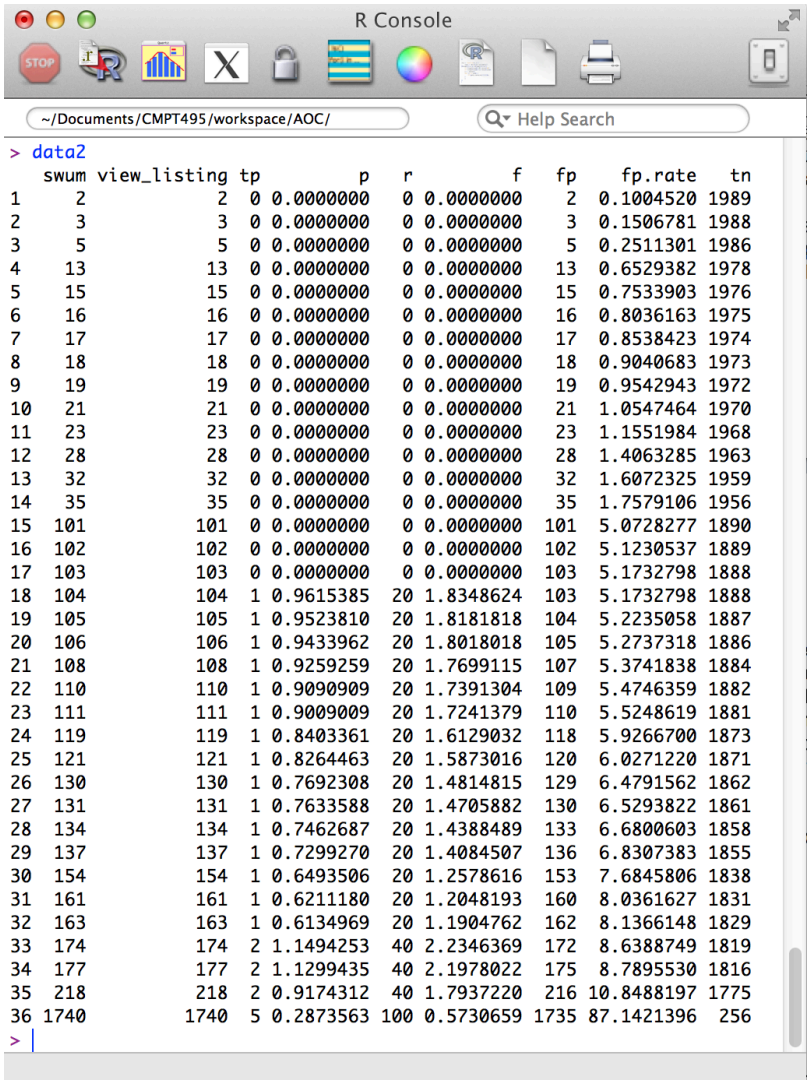
Here I create a data frame, prfdataframe, that will be used later on as the final data frame containing the technique, concern, query, precision, recall, f-measure, and MAP. The data and data2 data frames were created and used to store the information contained within each file that is listed in our files and files2 vectors. The read.table() function allows us to read in data line by line and use a delimiter, in this case the delimiting factor is simply white space (sep=””). It is important to note that I do not use the append() or rbind() function -- data and data2 will be overwritten on each iteration of the loop. They will exist once for each file that we read in.

I also use several arguments here; nrows=thresh in the data data frame will only read the first 10 lines so that we can calculate precision and recall (remember our threshold scalar?), and I use very specific regular expressions in conjunction with the substitute function when naming the column names - this way I can extract the name of the technique and query for each file. The data2 data frame also uses header=F and skip=1, which essentially says there is no header information for that file and to skip the first line (as the first line does contain a naming scheme for the columns of data, and I do not wish to use it and instead wish to import the data into my own column names).

Here is what one iteration of the loop will save to data and data2:



Note the name of the first column -- swum, and the name of the second column, view_listing. This is the file “view_listing” from the marked_search folder of “swum”.




```
> data2
   swum view_listing tp         p   r         f   fp    fp.rate   tn
1     2            2  0 0.0000000   0 0.0000000    2  0.1004520 1989
2     3            3  0 0.0000000   0 0.0000000    3  0.1506781 1988
3     5            5  0 0.0000000   0 0.0000000    5  0.2511301 1986
4    13           13  0 0.0000000   0 0.0000000   13  0.6529382 1978
5    15           15  0 0.0000000   0 0.0000000   15  0.7533903 1976
6    16           16  0 0.0000000   0 0.0000000   16  0.8036163 1975
7    17           17  0 0.0000000   0 0.0000000   17  0.8538423 1974
8    18           18  0 0.0000000   0 0.0000000   18  0.9040683 1973
9    19           19  0 0.0000000   0 0.0000000   19  0.9542943 1972
10   21           21  0 0.0000000   0 0.0000000   21  1.0547464 1970
11   23           23  0 0.0000000   0 0.0000000   23  1.1551984 1968
12   28           28  0 0.0000000   0 0.0000000   28  1.4063285 1963
13   32           32  0 0.0000000   0 0.0000000   32  1.6072325 1959
14   35           35  0 0.0000000   0 0.0000000   35  1.7579106 1956
15  101          101  0 0.0000000   0 0.0000000  101  5.0728277 1890
16  102          102  0 0.0000000   0 0.0000000  102  5.1230537 1889
17  103          103  0 0.0000000   0 0.0000000  103  5.1732798 1888
18  104          104  1 0.9615385  20 1.8348624  103  5.1732798 1888
19  105          105  1 0.9523810  20 1.8181818  104  5.2235058 1887
20  106          106  1 0.9433962  20 1.8018018  105  5.2737318 1886
21  108          108  1 0.9259259  20 1.7699115  107  5.3741838 1884
22  110          110  1 0.9090909  20 1.7391304  109  5.4746359 1882
23  111          111  1 0.9009009  20 1.7241379  110  5.5248619 1881
24  119          119  1 0.8403361  20 1.6129032  118  5.9266700 1873
25  121          121  1 0.8264463  20 1.5873016  120  6.0271220 1871
26  130          130  1 0.7692308  20 1.4814815  129  6.4791562 1862
27  131          131  1 0.7633588  20 1.4705882  130  6.5293822 1861
28  134          134  1 0.7462687  20 1.4388489  133  6.6800603 1858
29  137          137  1 0.7299270  20 1.4084507  136  6.8307383 1855
30  154          154  1 0.6493506  20 1.2578616  153  7.6845806 1838
31  161          161  1 0.6211180  20 1.2048193  160  8.0361627 1831
32  163          163  1 0.6134969  20 1.1904762  162  8.1366148 1829
33  174          174  2 1.1494253  40 2.2346369  172  8.6388749 1819
34  177          177  2 1.1299435  40 2.1978022  175  8.7895530 1816
35  218          218  2 0.9174312  40 1.7937220  216 10.8488197 1775
36 1740         1740  5 0.2873563 100 0.5730659 1735 87.1421396  256
> 
```

And here is data2, which as the column names suggest, read in the file “view_listing” from the prf_results folder within “swum”.

This loop is left open as I will calculate the precision, recall, f-measure, and MAP for each file as it is read in. Here is the rest of the loop:

```
#counters for P/R/F/MAP calculations
counter<-0
lastvalue<-0
rank<-0
map<-0
lastvalue<-0
```

Above I create several scalars with the initial value of 0. These will be used to calculate our precision, recall, f-measure, and MAP for each file. They must be created as scalars and equal to zero as they will need to reset with each iteration of the loop.

```
#loops through data looking for asterisks, keeps track of how many occur
for (i in 1:nrow(data)) {
```

```
 if (pmatch("*", data[1][i,], nomatch=FALSE, duplicates.ok=FALSE)==1) {
      counter<-counter + 1
      }
}
```

This loop will run through data (the one that only imports the first 10 lines) and will seek out and keep track of how many asterisks it finds by using the pmatch() function. I specify two arguments, nomatch=FALSE and duplicates.ok=FALSE, to ensure it doesn't return true if no asterisk is found. The pmatch() function simply looks for a pattern of characters within a string and will return "1" if it is found. Make note of the use of brackets to recall each individual element within the first column in data. I used numbers here since the column name will vary between files (remember the substitute function using the regular expressions?).

```
#calculates MAP via prf files
for (i in 1:nrow(data2)){
 if (data2$tp[i] != lastvalue) {
      map<-map + data2$p[i]
      lastvalue<-data2$tp[i]
      rank<-rank + 1
      }
}
```

This next loop will calculate MAP by looping through data2. Since we used a blocked rank design, I need to check to make sure that the value stored in data2$tp[i] (true positive) does not equal the previous one. This will add up all the values (as long as two or more identical tp numbers are not next to each other) and increment the rank variable.

```
#adds P/R/F/MAP to the data dataframe
data$P<-(counter/thresh)
 if (data$P[1] == "NaN" || data$P[1] == "Inf")
      data$P<-0
data$R<-(counter/data2$tp[length(data2$tp)])
 if (data$R[1] == "NaN" || data$R[1] == "Inf")
      data$R<-0
data$F<-(2*((data$P[1] * data$R[1])/(data$P[1] + data$R[1])))
 if (data$F[1] == "NaN" || data$F[1] == "Inf")
      data$F<-0
data$MAP<-(map/rank) / 100
 if (data$MAP[1] == "NaN" || data$MAP[1] == "Inf")
      data$MAP<-0
```

I can now add precision, recall, f-measure, and MAP information directly to data. Remember, data is overwritten with each iteration of the loop, so at the very end of this long loop I'll need to extract the data into a more permanent loop (remember the creation of prfdataframe?).

The if statements above will do the required calculations for precision, recall, f-measure, and MAP. I check to ensure that no "NaN" values or "Inf" values are added (for example, division by zero in R doesn't give an error, instead it returns "Inf").

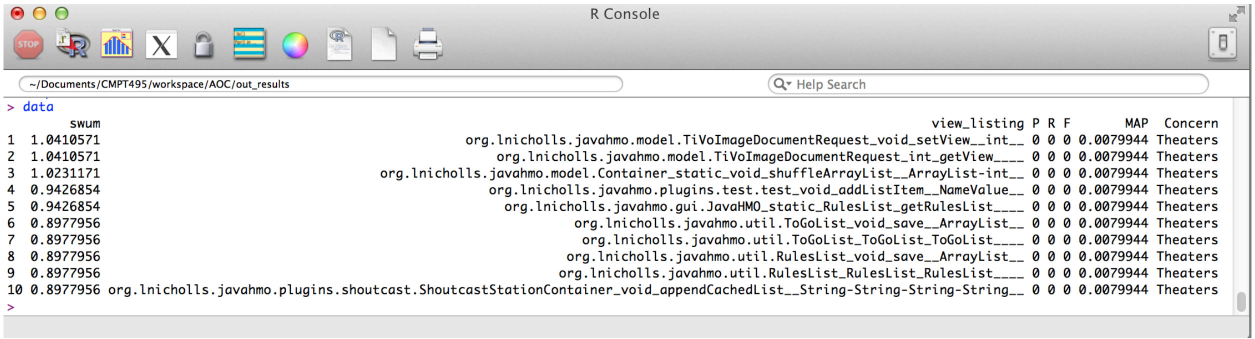
Note I use the length of data2$tp when calculating recall, since recall will be the amount of asterisks (relevant results) returned in the first 10 results divided by the amount of total asterisks. Using the length of the tp column in data2 (from the prf_results files) eliminates the need to read in the entirety of the marked_search files (which greatly reduces the amount of time needed to execute our code, as these files contain hundreds of lines).

I now have nearly all of the required information stored in data. All I need at this point is the concern that accompanies each query. I will now use many if statements in conjunction with grepl() and names() to look at the second column of data (the column that contains the returned queries with a variable name) to assign the concern name based on query:

```
#adds concern name to the dataframe
if (grepl("add_text", names(data[2])) == TRUE) {
 data$Concern <- "Textfield"
 }
if (grepl("create_textfield", names(data[2])) == TRUE) {
 data$Concern <- "Textfield"
 }
if (grepl("text_field_tool", names(data[2])) == TRUE) {
 data$Concern <- "Textfield"
 }
if (grepl("textfield", names(data[2])) == TRUE) {
 data$Concern <- "Textfield"
 }
if (grepl("textfieldReportElement", names(data[2])) == TRUE) {
 data$Concern <- "Textfield"
 }
if (grepl("IReportCompiler", names(data[2])) == TRUE) {
 data$Concern <- "Compile"
 }
if (grepl("compile", names(data[2])) == TRUE) {
 data$Concern <- "Compile"
 }
if (grepl("compile_report", names(data[2])) == TRUE) {
 data$Concern <- "Compile"
 }
if (grepl("add_auction", names(data[2])) == TRUE) {
 data$Concern <- "Add Auction"
 }
if (grepl("add_snipe", names(data[2])) == TRUE) {
 data$Concern <- "Set Snipe"
 }
if (grepl("do_you_wish_to_snipe", names(data[2])) == TRUE) {
 data$Concern <- "Set Snipe"
 }
if (grepl("set_snipe", names(data[2])) == TRUE) {
 data$Concern <- "Set Snipe"
 }
if (grepl("snipe", names(data[2])) == TRUE) {
 data$Concern <- "Set Snipe"
 }
if (grepl("sniping", names(data[2])) == TRUE) {
 data$Concern <- "Set Snipe"
 }
if (grepl("save", names(data[2])) == TRUE) {
 data$Concern <- "Save Auctions"
 }
if (grepl("save_auctions", names(data[2])) == TRUE) {
 data$Concern <- "Save Auctions"
```

```
 }
if (grepl("save_auction", names(data[2])) == TRUE) {
 data$Concern <- "Save Auctions"
 }
if (grepl("find", names(data[2])) == TRUE) {
 data$Concern <- "Theaters"
 }
if (grepl("get_theater", names(data[2])) == TRUE) {
 data$Concern <- "Theaters"
 }
if (grepl("movie", names(data[2])) == TRUE) {
 data$Concern <- "Theaters"
 }
if (grepl("theater", names(data[2])) == TRUE) {
 data$Concern <- "Theaters"
 }
if (grepl("view_listing", names(data[2])) == TRUE) {
 data$Concern <- "Theaters"
 }
if (grepl("search", names(data[2])) == TRUE) {
 data$Concern <- "Search"
 }
if (grepl("search_file", names(data[2])) == TRUE) {
 data$Concern <- "Search"
 }
if (grepl("open_file", names(data[2])) == TRUE) {
 data$Concern <- "Play"
 }
if (grepl("play", names(data[2])) == TRUE) {
 data$Concern <- "Play"
 }
if (grepl("play_track", names(data[2])) == TRUE) {
 data$Concern <- "Play"
 }
if (grepl("play_file", names(data[2])) == TRUE) {
 data$Concern <- "Play"
 }
if (grepl("start", names(data[2])) == TRUE) {
 data$Concern <- "Play"
 }
```

Ok, so we've made a lot of changes to the data data frame at this point. We've added several new columns, P, R, F, MAP, and Concern. Let's take a look at what data looks like now for this iteration:



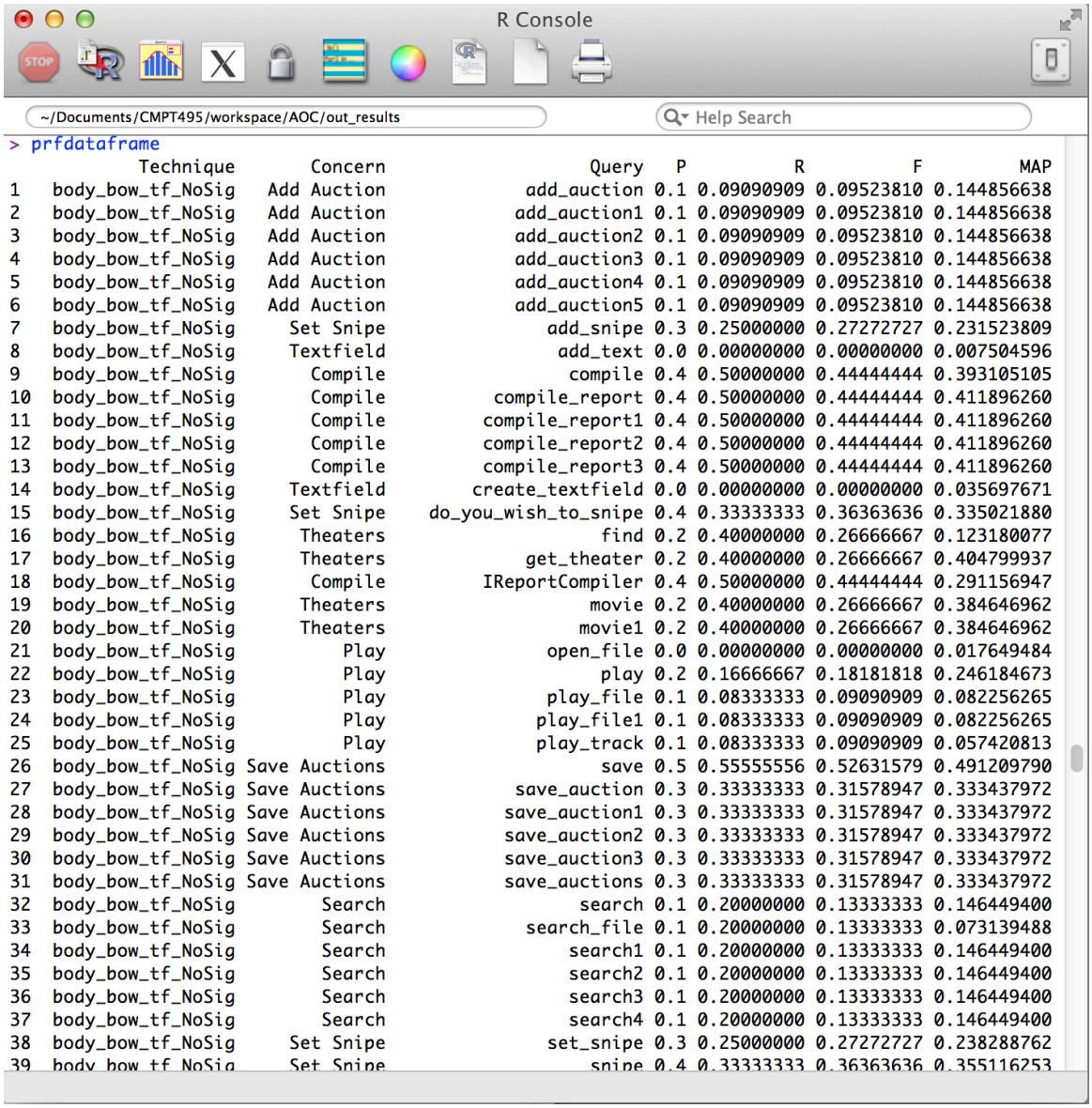
	swum	view_listing	P	R	F	MAP	Concern
1	1.0410571	org.lnicholls.javahmo.model.TiVoImageDocumentRequest_void_setView__int__	0	0	0	0.0079944	Theaters
2	1.0410571	org.lnicholls.javahmo.model.TiVoImageDocumentRequest_int_getView____	0	0	0	0.0079944	Theaters
3	1.0231171	org.lnicholls.javahmo.model.Container_static_void_shuffleArrayList__ArrayList-int__	0	0	0	0.0079944	Theaters
4	0.9426854	org.lnicholls.javahmo.plugins.test.test_void_addListItem__NameValue__	0	0	0	0.0079944	Theaters
5	0.9426854	org.lnicholls.javahmo.gui.JavaHMO_static_RulesList_getRulesList____	0	0	0	0.0079944	Theaters
6	0.8977956	org.lnicholls.javahmo.util.ToGoList_void_save__ArrayList__	0	0	0	0.0079944	Theaters
7	0.8977956	org.lnicholls.javahmo.util.ToGoList_ToGoList_ToGoList____	0	0	0	0.0079944	Theaters
8	0.8977956	org.lnicholls.javahmo.util.RulesList_void_save__ArrayList__	0	0	0	0.0079944	Theaters
9	0.8977956	org.lnicholls.javahmo.util.RulesList_RulesList_RulesList____	0	0	0	0.0079944	Theaters
10	0.8977956	org.lnicholls.javahmo.plugins.shoutcast.ShoutcastStationContainer_void_appendCachedList__String-String-String-String__	0	0	0	0.0079944	Theaters

We now have the specific technique (first column name), the query (second column name), the precision, recall, f-measure, and MAP for that set (P/R/F/MAP columns, all these values will always be the same), and the concern (last column entires). All that is left is to put these into a single data frame:

```
prfdataframe<-rbind(prfdataframe, data.frame("Technique"=names(data[1]),
 "Concern"=data$Concern[1],
 "Query"=names(data[2]), "P"=data$P[1], "R"=data$R[1], "F"=data$F[1], "MAP"=data
 $MAP[1]))

}
```

Here I finally make use of the prfdataframe that I created before this loop began. I use the rbind() function to add rows one by one -- one row for each iteration of the loop. I create the columns Technique and use the name of the first column in data, Concern which uses the first element in the Concern column of data, Query which uses the name of the second column in data, P/R/F/MAP which uses the first element in columns of the same name in data.

Here is a look at what the final prfdataframe will look like after the completion of the loop (all files read in):




```
> prfdataframe
          Technique       Concern                  Query   P         R         F         MAP
1   body_bow_tf_NoSig   Add Auction            add_auction 0.1 0.09090909 0.09523810 0.144856638
2   body_bow_tf_NoSig   Add Auction           add_auction1 0.1 0.09090909 0.09523810 0.144856638
3   body_bow_tf_NoSig   Add Auction           add_auction2 0.1 0.09090909 0.09523810 0.144856638
4   body_bow_tf_NoSig   Add Auction           add_auction3 0.1 0.09090909 0.09523810 0.144856638
5   body_bow_tf_NoSig   Add Auction           add_auction4 0.1 0.09090909 0.09523810 0.144856638
6   body_bow_tf_NoSig   Add Auction           add_auction5 0.1 0.09090909 0.09523810 0.144856638
7   body_bow_tf_NoSig     Set Snipe              add_snipe 0.3 0.25000000 0.27272727 0.231523809
8   body_bow_tf_NoSig     Textfield               add_text 0.0 0.00000000 0.00000000 0.007504596
9   body_bow_tf_NoSig       Compile                compile 0.4 0.50000000 0.44444444 0.393105105
10  body_bow_tf_NoSig       Compile         compile_report 0.4 0.50000000 0.44444444 0.411896260
11  body_bow_tf_NoSig       Compile        compile_report1 0.4 0.50000000 0.44444444 0.411896260
12  body_bow_tf_NoSig       Compile        compile_report2 0.4 0.50000000 0.44444444 0.411896260
13  body_bow_tf_NoSig       Compile        compile_report3 0.4 0.50000000 0.44444444 0.411896260
14  body_bow_tf_NoSig     Textfield       create_textfield 0.0 0.00000000 0.00000000 0.035697671
15  body_bow_tf_NoSig     Set Snipe    do_you_wish_to_snipe 0.4 0.33333333 0.36363636 0.335021880
16  body_bow_tf_NoSig      Theaters                    find 0.2 0.40000000 0.26666667 0.123180077
17  body_bow_tf_NoSig      Theaters             get_theater 0.2 0.40000000 0.26666667 0.404799937
18  body_bow_tf_NoSig       Compile        IReportCompiler 0.4 0.50000000 0.44444444 0.291156947
19  body_bow_tf_NoSig      Theaters                   movie 0.2 0.40000000 0.26666667 0.384646962
20  body_bow_tf_NoSig      Theaters                  movie1 0.2 0.40000000 0.26666667 0.384646962
21  body_bow_tf_NoSig          Play               open_file 0.0 0.00000000 0.00000000 0.017649484
22  body_bow_tf_NoSig          Play                    play 0.2 0.16666667 0.18181818 0.246184673
23  body_bow_tf_NoSig          Play               play_file 0.1 0.08333333 0.09090909 0.082256265
24  body_bow_tf_NoSig          Play              play_file1 0.1 0.08333333 0.09090909 0.082256265
25  body_bow_tf_NoSig          Play              play_track 0.1 0.08333333 0.09090909 0.057420813
26  body_bow_tf_NoSig Save Auctions                    save 0.5 0.55555556 0.52631579 0.491209790
27  body_bow_tf_NoSig Save Auctions            save_auction 0.3 0.33333333 0.31578947 0.333437972
28  body_bow_tf_NoSig Save Auctions           save_auction1 0.3 0.33333333 0.31578947 0.333437972
29  body_bow_tf_NoSig Save Auctions           save_auction2 0.3 0.33333333 0.31578947 0.333437972
30  body_bow_tf_NoSig Save Auctions           save_auction3 0.3 0.33333333 0.31578947 0.333437972
31  body_bow_tf_NoSig Save Auctions           save_auctions 0.3 0.33333333 0.31578947 0.333437972
32  body_bow_tf_NoSig        Search                  search 0.1 0.20000000 0.13333333 0.146449400
33  body_bow_tf_NoSig        Search             search_file 0.1 0.20000000 0.13333333 0.073139488
34  body_bow_tf_NoSig        Search                 search1 0.1 0.20000000 0.13333333 0.146449400
35  body_bow_tf_NoSig        Search                 search2 0.1 0.20000000 0.13333333 0.146449400
36  body_bow_tf_NoSig        Search                 search3 0.1 0.20000000 0.13333333 0.146449400
37  body_bow_tf_NoSig        Search                 search4 0.1 0.20000000 0.13333333 0.146449400
38  body_bow_tf_NoSig     Set Snipe               set_snipe 0.3 0.25000000 0.27272727 0.238288762
39  body_bow_tf_NoSig     Set Snipe                   snipe 0.4 0.33333333 0.36363636 0.355116253
```

Nice, neat, and tidy!

Now that we all of our data contained within a single object, the next thing we need to do is to begin a statistical analysis. This will be accomplished by taking a quick look at the summary statistics for P/R/F/MAP for each technique, as well as creating boxplots of each technique vs MAP (since we're looking for differences in the mean of MAP for each technique). Finally, we will run an analysis of variance test (aov) and a Tukey HSD test to determine statistical significance between the MAP means of each technique.

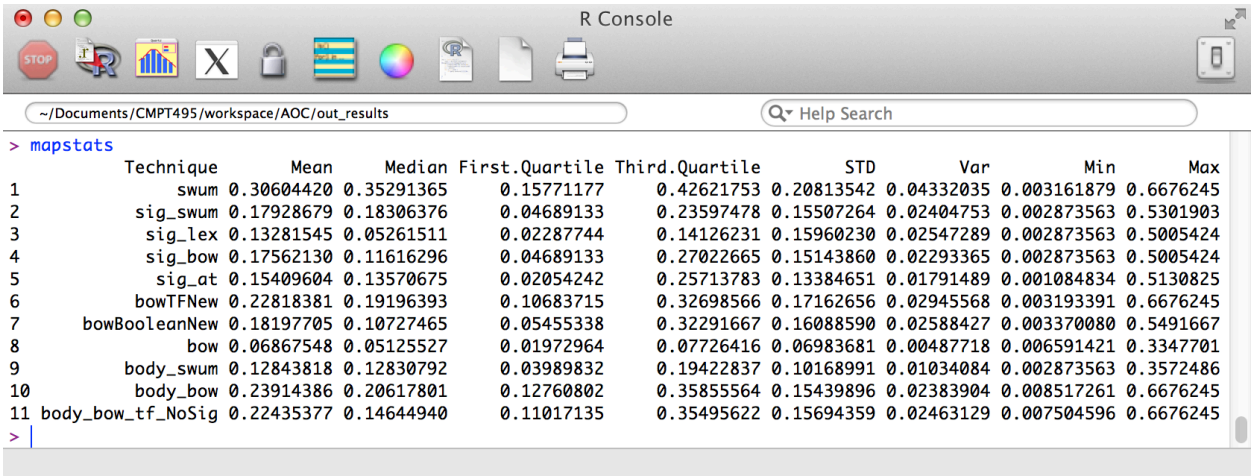
The next bit of code:

```
#empty data frame for p/r/f/map stats
mapstats<-data.frame()
pstats<-data.frame()
rstats<-data.frame()
fstats<-data.frame()
```

```
#calculates mean/std/var/median/quartiles for p/r/f/map data
for (i in c("swum", "sig_swum", "sig_lex", "sig_bow", "sig_at",
 "bowTFNew","bowBooleanNew", "bow", "body_swum", "body_bow", "body_bow_tf_NoSig")) {

     ci<-subset(prfdataframe, prfdataframe[1]==i)

     mapstats<-rbind(mapstats, data.frame("Technique"=i, "Mean"=mean(ci$MAP),
 "Median"=quantile(ci$MAP,.50), "First.Quartile"=quantile(ci$MAP,.25),
 "Third.Quartile"=quantile(ci$MAP,.75),"STD"=sd(ci$MAP), "Var"=var(ci$MAP),
 "Min"=min(ci$MAP), "Max"=max(ci$MAP), row.names=NULL))

     pstats<-rbind(pstats, data.frame("Technique"=i, "Mean"=mean(ci$P),
 "Median"=quantile(ci$P,.50),"First.Quartile"=quantile(ci$P,.25),
 "Third.Quartile"=quantile(ci$P,.75), "STD"=sd(ci$P),"Var"=var(ci$P), "Min"=min(ci$P),
 "Max"=max(ci$P), row.names=NULL))

     rstats<-rbind(rstats, data.frame("Technique"=i, "Mean"=mean(ci$R),
 "Median"=quantile(ci$R,.50),"First.Quartile"=quantile(ci$R,.25),
 "Third.Quartile"=quantile(ci$R,.75), "STD"=sd(ci$R),"Var"=var(ci$R), "Min"=min(ci$R),
 "Max"=max(ci$R), row.names=NULL))

     fstats<-rbind(fstats, data.frame("Technique"=i, "Mean"=mean(ci$F),
 "Median"=quantile(ci$F,.50),"First.Quartile"=quantile(ci$F,.25),
 "Third.Quartile"=quantile(ci$F,.75), "STD"=sd(ci$F),"Var"=var(ci$F), "Min"=min(ci$F),
 "Max"=max(ci$F), row.names=NULL))
}
```

The above code begins with the creation of several empty data frames. As I will be using rbind(), I need to create them as empty data frames before the loop begins. The loop will cycle through a list of values (swum, sig_swum, etc) and create a subset of those values from prfdataframe. The subset function takes a data frame and a logical statement, in this instance we look for the technique names and create a data frame containing all the entries with the same technique name.

Next we fill up the mapstats, pstats, rstats, and fstats data frames with summary statistics information. Mean, median, quartiles, etc. Our subset data frame, ci, will be overwritten on each iteration. Here is what mapstats looks like at the end of the loop:




```
> mapstats
          Technique       Mean     Median First.Quartile Third.Quartile        STD        Var         Min       Max
1              swum 0.30604420 0.35291365     0.15771177     0.42621753 0.20813542 0.04332035 0.003161879 0.6676245
2          sig_swum 0.17928679 0.18306376     0.04689133     0.23597478 0.15507264 0.02404753 0.002873563 0.5301903
3           sig_lex 0.13281545 0.05261511     0.02287744     0.14126231 0.15960230 0.02547289 0.002873563 0.5005424
4           sig_bow 0.17562130 0.11616296     0.04689133     0.27022665 0.15143860 0.02293365 0.002873563 0.5005424
5            sig_at 0.15409604 0.13570675     0.02054242     0.25713783 0.13384651 0.01791489 0.001084834 0.5130825
6          bowTFNew 0.22818381 0.19196393     0.10683715     0.32698566 0.17162656 0.02945568 0.003193391 0.6676245
7      bowBooleanNew 0.18197705 0.10727465     0.05455338     0.32291667 0.16088590 0.02588427 0.003370080 0.5491667
8                bow 0.06867548 0.05125527     0.01972964     0.07726416 0.06983681 0.00487718 0.006591421 0.3347701
9         body_swum 0.12843818 0.12830792     0.03989832     0.19422837 0.10168991 0.01034084 0.002873563 0.3572486
10         body_bow 0.23914386 0.20617801     0.12760802     0.35855564 0.15439896 0.02383904 0.008517261 0.6676245
11 body_bow_tf_NoSig 0.22435377 0.14644940     0.11017135     0.35495622 0.15694359 0.02463129 0.007504596 0.6676245
>
```

A statistical summary for each technique's MAP values across all concerns. This will be completed for precision, recall, and f-measure as well. These summaries need to be printed to a file, which we will do next:

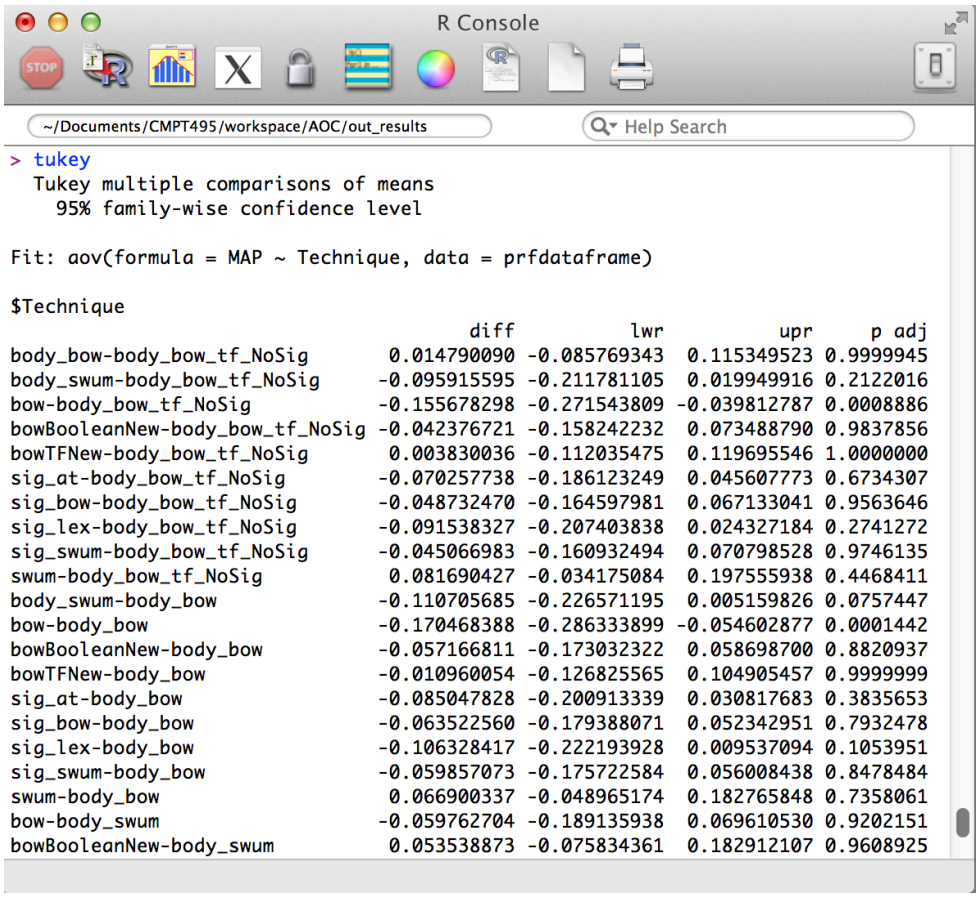
```
#prints out summary statistics and save them to files
sink(paste(outputdir, "/precisionStats.txt", sep=""))
print("-----------precision summary-------------")
pstats
sink()

sink(paste(outputdir, "/recallStats.txt", sep=""))
print("------------recall summary--------------")
rstats
sink()

sink(paste(outputdir, "/fmeasureStats.txt", sep=""))
print("-----------f-measure summary-------------")
fstats
sink()

sink(paste(outputdir, "/mapStats.txt", sep=""))
print("-------------map summary---------------")
mapstats
sink()
```

We use the sink() function and paste() function in conjunction to append the name of the file to the end of the string held in outputdir (one of the first lines written in this code!). The paste() function takes arguments in this format: paste(FIRST, CONCATENATE_THIS, sep=WHAT_TO_SEPARATE_BY). So we add the specific file name to the end of the value stored in outputdir, without any spaces in between. We then make a call to the objects, and instead of printing to the console they will print to the specified files. Finally, we close the device by calling sink() again. Failure to close sink() will keep the device open, which means any output intended for the console from here on out will instead be sent to the last text file.

Now it's time for an analysis of variance and Tukey HSD test:

```
sink(paste(outputdir, "/TukeyResults.txt", sep=""))
print("-------------tukeyhsd/aov--------------")
analysis<-aov(formula=MAP~Technique, data=prfdataframe)
summary(analysis)
tukey<-TukeyHSD(analysis)
print(tukey)
```

Again, we save all of these results to the file "TukeyResults.txt". We use the aov() function to perform the analysis of variance with the formula MAP~Technique. We do a summary of the aov() object, and finally run the Tukey HSD test on it. Note that sink() is left open. This is what our tukey object looks like:




```
> tukey
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = MAP ~ Technique, data = prfdataframe)

$Technique
                                          diff          lwr          upr     p adj
body_bow-body_bow_tf_NoSig          0.014790090 -0.085769343  0.115349523 0.9999945
body_swum-body_bow_tf_NoSig        -0.095915595 -0.211781105  0.019949916 0.2122016
bow-body_bow_tf_NoSig              -0.155678298 -0.271543809 -0.039812787 0.0008886
bowBooleanNew-body_bow_tf_NoSig    -0.042376721 -0.158242232  0.073488790 0.9837856
bowTFNew-body_bow_tf_NoSig          0.003830036 -0.112035475  0.119695546 1.0000000
sig_at-body_bow_tf_NoSig           -0.070257738 -0.186123249  0.045607773 0.6734307
sig_bow-body_bow_tf_NoSig          -0.048732470 -0.164597981  0.067133041 0.9563646
sig_lex-body_bow_tf_NoSig          -0.091538327 -0.207403838  0.024327184 0.2741272
sig_swum-body_bow_tf_NoSig         -0.045066983 -0.160932494  0.070798528 0.9746135
swum-body_bow_tf_NoSig              0.081690427 -0.034175084  0.197555938 0.4468411
body_swum-body_bow                 -0.110705685 -0.226571195  0.005159826 0.0757447
bow-body_bow                       -0.170468388 -0.286333899 -0.054602877 0.0001442
bowBooleanNew-body_bow             -0.057166811 -0.173032322  0.058698700 0.8820937
bowTFNew-body_bow                  -0.010960054 -0.126825565  0.104905457 0.9999999
sig_at-body_bow                    -0.085047828 -0.200913339  0.030817683 0.3835653
sig_bow-body_bow                   -0.063522560 -0.179388071  0.052342951 0.7932478
sig_lex-body_bow                   -0.106328417 -0.222193928  0.009537094 0.1053951
sig_swum-body_bow                  -0.059857073 -0.175722584  0.056008438 0.8478484
swum-body_bow                       0.066900337 -0.048965174  0.182765848 0.7358061
bow-body_swum                      -0.059762704 -0.189135938  0.069610530 0.9202151
bowBooleanNew-body_swum             0.053538873 -0.075834361  0.182912107 0.9608925
```

We will need to make use of tukey$Technique, as this object is a list that contains an entire data frame as a column (namely, the $Technique column). We create a new data frame from $Technique:

```
tukeyframe<-data.frame(tukey$Technique)

sigresults<-data.frame()
for (i in 1:length(tukeyframe[,1]))
if (tukeyframe$p.adj[i] < .05)
 sigresults<-rbind(sigresults, tukeyframe[i,])

print("-------------tukey significant results-------------")
print(sigresults)
sink()
```

We also create a data frame called sigresults. We loop through tukeyframe and check the “p.adj” column for values that are less than .05, as these values are statistically significant. We append these values to the end of sigresults, using rbind(). Finally, we print the results and close the device by using sink(). Our TukeyResults.txt file should now contain all of the TukeyHSD test results, as well as a list of the values that are statistically significant. Here is a look at that:

TukeyResults.txt

```
[1] "-------------tukeyhsd/aov--------------"
             Df Sum Sq Mean Sq F value   Pr(>F)
Technique    10  1.285 0.12849   5.559 1.07e-07 ***
Residuals   346  7.997 0.02311
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = MAP ~ Technique, data = prfdataframe)

$Technique
                                          diff          lwr          upr     p adj
body_bow-body_bow_tf_NoSig        0.014790090 -0.085769343  0.115349523 0.9999945
body_swum-body_bow_tf_NoSig      -0.095915595 -0.211781105  0.019949916 0.2122016
bow-body_bow_tf_NoSig            -0.155678298 -0.271543809 -0.039812787 0.0008886
bowBooleanNew-body_bow_tf_NoSig  -0.042376721 -0.158242232  0.073488790 0.9837856
bowTFNew-body_bow_tf_NoSig        0.003830036 -0.112035475  0.119695546 1.0000000
sig_at-body_bow_tf_NoSig         -0.070257738 -0.186123249  0.045607773 0.6734307
sig_bow-body_bow_tf_NoSig        -0.048732470 -0.164597981  0.067133041 0.9563646
sig_lex-body_bow_tf_NoSig        -0.091538327 -0.207403838  0.024327184 0.2741272
sig_swum-body_bow_tf_NoSig       -0.045066983 -0.160932494  0.070798528 0.9746135
swum-body_bow_tf_NoSig            0.081690427 -0.034175084  0.197555938 0.4468411
body_swum-body_bow               -0.110705685 -0.226571195  0.005159826 0.0757447
bow-body_bow                     -0.170468388 -0.286333899 -0.054602877 0.0001442
bowBooleanNew-body_bow           -0.057166811 -0.173032322  0.058698700 0.8820937
bowTFNew-body_bow                -0.010960054 -0.126825565  0.104905457 0.9999999
sig_at-body_bow                  -0.085047828 -0.200913339  0.030817683 0.3835653
sig_bow-body_bow                 -0.063522560 -0.179388071  0.052342951 0.7932478
sig_lex-body_bow                 -0.106328417 -0.222193928  0.009537094 0.1053951
sig_swum-body_bow                -0.059857073 -0.175722584  0.056008438 0.8478484
swum-body_bow                     0.066900337 -0.048965174  0.182765848 0.7358061
bow-body_swum                    -0.059762704 -0.189135938  0.069610530 0.9202151
bowBooleanNew-body_swum           0.053538873 -0.075834361  0.182912107 0.9608925
bowTFNew-body_swum                0.099745630 -0.029627604  0.229118864 0.3091307
sig_at-body_swum                  0.025657857 -0.103715377  0.155031091 0.9999087
sig_bow-body_swum                 0.047183124 -0.082190110  0.176556358 0.9841238
sig_lex-body_swum                 0.004377268 -0.124995966  0.133750502 1.0000000
sig_swum-body_swum                0.050848611 -0.078524623  0.180221845 0.9726602
swum-body_swum                    0.177606022  0.048232788  0.306979256 0.0005944
bowBooleanNew-bow                 0.113301577 -0.016071657  0.242674811 0.1483514
bowTFNew-bow                      0.159508334  0.030135100  0.288881568 0.0037681
sig_at-bow                        0.085420561 -0.043952673  0.214793795 0.5497177
sig_bow-bow                       0.106945828 -0.022427406  0.236319062 0.2139695
sig_lex-bow                       0.064139971 -0.065233263  0.193513206 0.8788131
sig_swum-bow                      0.110611315 -0.018761919  0.239984549 0.1740318
swum-bow                          0.237368725  0.107995491  0.366741959 0.0000004
bowTFNew-bowBooleanNew            0.046206757 -0.083166477  0.175579991 0.9864255
sig_at-bowBooleanNew             -0.027881016 -0.157254251  0.101492218 0.9998052
sig_bow-bowBooleanNew            -0.006355749 -0.135728983  0.123017485 1.0000000
sig_lex-bowBooleanNew            -0.049161606 -0.178534840  0.080211628 0.9785310
sig_swum-bowBooleanNew           -0.002690262 -0.132063496  0.126682972 1.0000000
swum-bowBooleanNew                0.124067148 -0.005306086  0.253440382 0.0733728
sig_at-bowTFNew                  -0.074087773 -0.203461007  0.055285461 0.7455310
sig_bow-bowTFNew                 -0.052562506 -0.181935740  0.076810728 0.9655277
sig_lex-bowTFNew                 -0.095368362 -0.224741597  0.034004872 0.3769532
sig_swum-bowTFNew                -0.048897019 -0.178270253  0.080476215 0.9793561
swum-bowTFNew                     0.077860391 -0.051512843  0.207233626 0.6834931
sig_bow-sig_at                    0.021525267 -0.107847967  0.150898501 0.9999822
sig_lex-sig_at                   -0.021280589 -0.150653823  0.108092645 0.9999840
sig_swum-sig_at                   0.025190754 -0.104182480  0.154563989 0.9999229
swum-sig_at                       0.151948165  0.022574931  0.281321399 0.0076598
sig_lex-sig_bow                  -0.042805857 -0.172179091  0.086567378 0.9924656
sig_swum-sig_bow                  0.003665487 -0.125707747  0.133038721 1.0000000
swum-sig_bow                      0.130422897  0.001049663  0.259796131 0.0462203
sig_swum-sig_lex                  0.046471344 -0.082901890  0.175844578 0.9858295
swum-sig_lex                      0.173228754  0.043855520  0.302601988 0.0009463
swum-sig_swum                     0.126757410 -0.002615824  0.256130644 0.0605826

[1] "-------------tukey significant results-------------"
                          diff          lwr         upr        p.adj
bow-body_bow_tf_NoSig -0.1556783 -0.271543809 -0.03981279 8.886112e-04
bow-body_bow          -0.1704684 -0.286333899 -0.05460288 1.441631e-04
swum-body_swum         0.1776060  0.048232788  0.30697926 5.943522e-04
bowTFNew-bow           0.1595083  0.030135100  0.28888157 3.768149e-03
swum-bow               0.2373687  0.107995491  0.36674196 3.678988e-07
swum-sig_at            0.1519482  0.022574931  0.28132140 7.659796e-03
swum-sig_bow           0.1304229  0.001049663  0.25979613 4.622032e-02
swum-sig_lex           0.1732288  0.043855520  0.30260199 9.463387e-04
```

We're starting to see some results here! In a formal statistical analysis we would do the analysis of variance and Tukey HSD tests after we create the boxplots. Since I didn't write the code in that order, we don't do that. For the purpose of this paper, that's okay.

Now for some boxplots! We're going to print all these boxplots to PDF files. We'll use a loop, as we will need many boxplot combinations. We're going to start with a loop that plots the means of P/R/F/MAP for each technique, and for each concern:

```
#creates directory to store boxplots
dir.create(paste(outputdir, "/Boxplots",sep=""), showWarnings = FALSE)

#loop to create boxplots (map/p/r/f vs technique and concern)

for (i in 4:7) {
 for (x in 1:2) {
     prfdataframe[,x]<-with(prfdataframe, reorder(prfdataframe[,x], prfdataframe[[i]],
 mean))
     name<-paste(outputdir, "/Boxplots/", names(prfdataframe[i]), "_",
          names(prfdataframe[x]), "boxplot.pdf", sep="")
     main<-paste(names(prfdataframe[i]),"~", names(prfdataframe[x]), sep="")
     pdf(name)
     par(mar=c(9,4,2,1))
     boxplot(prfdataframe[[i]]~prfdataframe[[x]], data=prfdataframe,
 ylab=names(prfdataframe[i]), col="lightgray",
          las=2, main=main, cex.axis=1)
     means<-tapply(prfdataframe[[i]], prfdataframe[,x], mean)
     points(means, col="red")
     dev.off()
     }
 }
```

First order of business was to create a new directory, /Boxplots. The dir.create() function by nature will spam the console with warning messages that the directory doesn't exist and that it created it. Since we don't need to see this stuff, we use showWarnings=FALSE to shut it up. There is a lot going on within this loop, so I'll go through it step by step.

I use an outer loop from 4 to 7. These numbers will coincide with the column numbers for precision, recall, f-measure, and MAP within prfdataframe (our large data frame with all the good stuff). The inner loop will cycle from 1 to 2, numbers that coincide with the technique and concern columns. The goal is to make a boxplot for every P/R/F/MAP/technique/concern combination.

The very first thing we do is reorder the data within prfdataframe by the mean of P/R/F/MAP (whichever our loop is on at the time). You MUST order your data prior to entry into the boxplot if you intend to have it in some specific order.

I make an object called name which will use the name of the outer, and the name of the inner loop (for example, P_<technique>) and use paste to splice it all together. This will be the name of our boxplot PDF for that specific combination.

I also make another object called main, which will also extract names for use as the primary name of the boxplot. Next I open up a pdf device with the argument of name, which will direct all output to the path we created with the name object.

I use the par() function in conjunction with the mar() function. The par() function is used to specify parameters to be used with the boxplot, and it must be used prior to the creation of the boxplot (if you wish to change parameters, that is). In this case, we wish to adjust the margins using the mar() function.

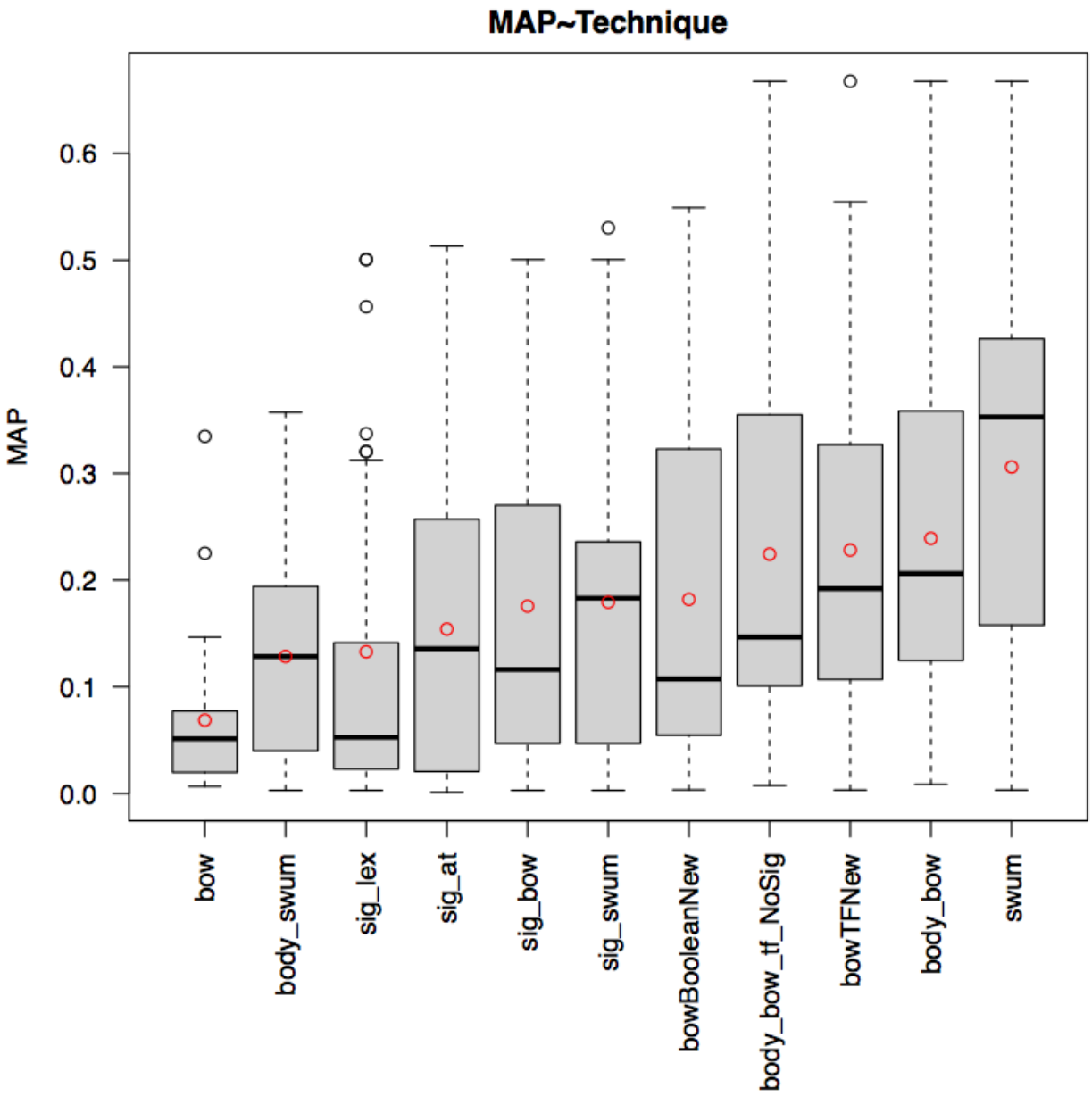
Now for the creation of the boxplot itself -- I'm going to re-copy that bit to explain it a little better:

```
boxplot(prfdataframe[[i]]~prfdataframe[[x]], data=prfdataframe,
 ylab=names(prfdataframe[i]), col="lightgray",las=2, main=main, cex.axis=1)
```

The boxplot() function will need a formula passed to it, in this case we're using prfdataframe[[i]]~prfdataframe[[x]], which will essentially translate to P/R/F/MAP ~ Technique/Concern via our loops. We specify where this data is coming from with data=prfdataframe, and assign the y axis labels by using the names function. So when we're on i=4, which will be precision, our y label will simply be precision. Finally, we define a color for the boxplots with col="lightgray", rotate the x axis labels with las=2 (otherwise they wouldn't fit!), give a name to our boxplot with main=main (our object created earlier), and define the size of the x axis font with cex.axis=1 (this is not necessary but the option exists).

The remaining bits of code from the loop will use the tapply() function to create a new object called "means", which will will use as points on our boxplots. The tapply() function is used to apply a function to a specified set of data. In this case, we pass the arguments of the numeric data in P/R/F/MAP along with the name of the technique/concern, and calculate the means. This is then stored in our means object. It will be stored in order of mean, since our prfdataframe is now sorted by mean.

Finally, we use the points() function to draw points on our boxplot. We use col="red" so they will show up as red circles. Here is a look at one of our boxplots:



Arranged in order of increasing mean, with the name of the numeric portion on the Y axis, and the techniques on the X axis. Looks like swum has quite the lead!

Now we want to create some boxplots for each P/R/F/MAP and Technique combination, but using only one concern at a time.

```
#creates directory to store boxplots
dir.create(paste(outputdir, "/BoxplotsByConcern", sep=""), showWarnings = FALSE)

#loop to create boxplots (map/p/r/f vs technique by one concern at a time)
for (c in c("Textfield", "Compile", "Add Auction", "Set Snipe", "Save Auctions",
 "Theaters", "Search", "Play")) {
 concernframe<-subset(prfdataframe, prfdataframe$Concern == c)
```

```
for (i in 4:7) {
     concernframe[,1]<-with(concernframe, reorder(concernframe[,1], concernframe[[i]],
mean))
     name<-paste(outputdir, "/BoxplotsByConcern/", names(concernframe[i]), "_",
          names(concernframe[1]), "_", c,"_","boxplot.pdf", sep="")
     main<-paste(names(concernframe[i]),"~", names(concernframe[1]), "(", c, ")",
sep="")
     pdf(name)
     par(mar=c(9,4,2,1))
     boxplot(concernframe[[i]]~concernframe[[1]], data=concernframe,
ylab=names(concernframe[i]),
          col="lightgray", las=2, main=main, cex.axis=1)
     means<-tapply(concernframe[[i]], concernframe[,1], mean)
     points(means, col="red")
     dev.off()
     }
}
```

In the above code we create a subset based on concern names, and create boxplots in the same way we did with our first loop. This time around, however, we only use a single loop since we don't need to bounce back and forth from technique and concern. We also adjust the name object and main object a little bit to include which concern we're currently looking at.

Our last few loops will be rather similar to the last boxplot loop, except this time around we will be subsetting parts of our prfdataframe so we can create boxplots based on being sig-only, body-only, and a combination of the two.

```
#direcory for paired boxplots
dir.create(paste(outputdir, "/PairedBoxplots", sep=""), showWarnings = FALSE)

#Sig-Only box plot
sigframe<-subset(prfdataframe, prfdataframe$Technique=="sig_swum" | prfdataframe
 $Technique == "sig_at" | prfdataframe$Technique == "sig_bow" | prfdataframe$Technique
 == "sig_lex")
 for (i in 4:7) {
     sigframe[,1]<-with(sigframe, reorder(sigframe[,1], sigframe[[i]], mean))
     name<-paste(outputdir, "/PairedBoxplots/", "SigOnly_", names(sigframe[i]), "_",
     names(sigframe[1]), "_","boxplot.pdf", sep="")
     main<-paste(names(sigframe[i]),"~", names(sigframe[1]), sep="")
     pdf(name)
     par(mar=c(9,4,2,1))
     boxplot(sigframe[[i]]~droplevels(sigframe[[1]]), data=sigframe,
     ylab=names(sigframe[i]), col="lightgray", las=2, main=main, cex.axis=1)
     means<-tapply(sigframe[[i]], sigframe[,1], mean)
     points(means, col="red")
     dev.off()
 }
```

Our sig-only loop above creates a subset named sigframe with the techniques sig_swum, sig_at, sig_box, and sig_lex (note the logical statements while subsetting). We only make boxplots based on technique, so we have a single loop instead of two (no need for concern boxplots here).

```
#Body-Only box plot
```

```
bodyframe<-subset(prfdataframe, prfdataframe$Technique=="body_swum" | prfdataframe
 $Technique == "bow" | prfdataframe$Technique == "body_bow_tf_NoSig")
 for (i in 4:7) {
      bodyframe[,1]<-with(bodyframe, reorder(bodyframe[,1], bodyframe[[i]], mean))
      name<-paste(outputdir, "/PairedBoxplots/", "BodyOnly_", names(bodyframe[i]), "_",
      names(bodyframe[1]), "_","boxplot.pdf", sep="")
      main<-paste(names(bodyframe[i]),"~", names(bodyframe[1]), sep="")
      pdf(name)
      par(mar=c(9,4,2,1))
      boxplot(bodyframe[[i]]~droplevels(bodyframe[[1]]), data=bodyframe,
      ylab=names(bodyframe[i]), col="lightgray", las=2, main=main, cex.axis=1)
      means<-tapply(bodyframe[[i]], bodyframe[,1], mean)
      points(means, col="red")
      dev.off()
 }
```

Our body-only loop above, much like our sig-only loop, creates a subset from prfdataframe with the techniques body_swum, bow, and body_bow_tf_NoSig.

```
#Both box plot
bothframe<-subset(prfdataframe, prfdataframe$Technique=="swum" | prfdataframe
 $Technique == "bowBooleanNew"
 | prfdataframe$Technique == "bowTFNew" | prfdataframe$Technique == "body_bow")
 for (i in 4:7) {
      bothframe[,1]<-with(bothframe, reorder(bothframe[,1], bothframe[[i]], mean))
      name<-paste(outputdir, "/PairedBoxplots/", "Both_", names(bothframe[i]), "_",
      names(bothframe[1]), "_","boxplot.pdf", sep="")
      main<-paste(names(bothframe[i]),"~", names(bothframe[1]), sep="")
      pdf(name)
      par(mar=c(9,4,2,1))
      boxplot(bothframe[[i]]~droplevels(bothframe[[1]]), data=bothframe,
      ylab=names(bothframe[i]), col="lightgray", las=2, main=main, cex.axis=1)
      means<-tapply(bothframe[[i]], bothframe[,1], mean)
      points(means, col="red")
      dev.off()
 }
```

Lastly, our loop above creates boxplots based on the techniques that are a combination of sig and body. These are the swum, bowBooleanNew, bowTFNew, and body_bow techniques.

That's it! Our finished program will now read in all required data, sort it appropriately, calculate the appropriate statistics we need, and output everything to a text file or a PDF.

Section 6: Links

Just some links I've acquired during this independent study.

R multiple comparisons
http://www.agr.kuleuven.ac.be/vakken/statisticsbyR/ANOVAbyRr/multiplecomp.htm

Quick-R ANOVA/MANOVA
http://www.statmethods.net/stats/anova.html

Quick-R General
http://www.statmethods.net/index.html

Multivariate regression
http://www.psych.yorku.ca/lab/psy6140/lectures/MultivariateRegression2x2.pdf

How to input data into R
http://www.ats.ucla.edu/stat/r/faq/inputdata_R.htm

R: List files in a directory/folder
http://stat.ethz.ch/R-manual/R-devel/library/base/html/list.files.html

A brief introduction to R
http://xweb.geos.ed.ac.uk/~hcp/r_notes/r_notes.html

Average Precision/Mean Average Precision wikipedia
http://en.wikipedia.org/wiki/Information_retrieval#F-measure

More on precision/recall/f-measure/ap/map
http://web.cecs.pdx.edu/~maier/cs510iri/IR_Lectures/CS_510iri_Lecture8RelevanceEvaluation-revised.pdf

The R apply function: a tutorial with examples
http://www.r-bloggers.com/the-r-apply-function-—-a-tutorial-with-examples/

Merging data frames
http://stat.ethz.ch/R-manual/R-patched/library/base/html/merge.html

More on merging data frames
http://rwiki.sciviews.org/doku.php?id=tips:data-frames:merge