Spencer Phillips

998887568

12/9/15

STA141 Assignment 6

Starting off with this assignment I started up the RCurl and XML packages and then called in page 3 and the last page of the tagged r StackOverflow page in order to have pages that would include different issues that I could work with.

url = "https://stackoverflow.com/questions/tagged/r?page=3&sort=newest&pagesize=50"

lasturl = "https://stackoverflow.com/questions/tagged/r?page=2335&sort=newest&pagesize=50"

blerg = getURLContent(url, verbose = TRUE)

lastblerg = getURLContent(lasturl, verbose = TRUE)

lasthtml = htmlParse(lastblerg)

html = htmlParse(blerg)

I then proceeded to make functions to grab each of the variables I needed. First was the function to get the poster of the post which had a call to "//div[@class = 'user-details']", then for looped to check for ".//a/text()". It considered any null cases to replace with NA and otherwise placed into itself again as its XMLValue.

For getting the time of the post, I had a call to "//div[@class = 'user-action-time']/span/@title" and then put the values into a dataframe.

For the name of the post I called to "//div[@class = 'summary']/h3/a/text()" then unlisted the values and ran an sapply to get the xmlValues of all of the values.

To get the reputation of the poster, I called to "//div[@class = 'user-details']" then for looped to get ".//div[@class = '-flair']/span[@class = 'reputation-score']/text()" and then took the xmlValue of that. I went through and made into NA those that were null and for those that ended with k, I got the full number by explicitly checking for ".//div[@class = '-flair']/span/@title" to get the real number since the real number for those with a k was hidden in the title (each was unique) then gsubbing it to take out any extra stuff. I also removed any commas that were present as well for those that were too small for a k.

For views, I checked for "//div[@class = 'statscontainer']" and then in a for loop checked for the following ".//div[@class = 'views ']/@title| .//div[@class = 'views warm']/@title | .//div[@class = 'views hot']/@title"). Since there were 3 different classes for the views depending on how many times the question had been viewed. I then took the gsub of the view numbers so there were no commas present as well.

For the number of answers for the post, I had to consider "//div[@class = 'status unanswered']/strong/text() |//div[@class = 'status answered']/strong/text() |//div[@class = 'status answered-accepted']/strong/text()" in order to get all values that I wanted There were different classes for the value being answered, unanswered, and having an accepted answer. After checking for all of these, I ran a sapply to get the xmlValues of each of them.

For the score of the post I ran the statement "//span[@class = 'vote-count-post ']/strong/text()" followed by an sapply for the xmlValues.

For the URL of the post I used the argument "//div[@class = 'summary']/h3/a/@href" then ran an sapply for the values I got, which put each value into a getRelativeURL argument to get the complete URLs.

For the post ID, I used the html argument "//div[@class = 'question-summary']/@id" and then ran a gsub to only get the numbers of the post ID since the id had other values that I did not want.

Finally to get the tags of each value, I used the argument "//div[@class = 'summary']" to get the overarching parts. I then looped over each value and using xpathsapply used ".//div/a[@class = 'post-tag']" with xmlValue. I then collapsed each one together with a semicolon and a space in order to keep it neat.

All of these functions returned data.frame objects as it would make it easier to merge them all together later on. I tended to use text() since it instantly got me the data that I needed and allowed me to make sure it was the data I wanted before running it through xmlValue() to make it into character values.

I then created a function that would get the next page in order to continue the loop and set in a test that would return NULL if there were no other pages left. This used the argument "//div[@class = 'pager fl']/a[@rel = 'next']/@href" to get the url for the next page. This then put together the part that was given with the stackoverflow.com part to make a new URL.

Then, I made a function that would take in all of the functions I made that would grab the variables and put them together into one large dataset together.

I merged all of these functions into one called StackOverflow() where it would loop until it ran out of pages or hit a page limit cap getting new pages and adding them to the overarching dataset that would be built until the page limit was hit or it ran out of pages to scrape. It then renamed the columns to better names and returned the dataframe

I chose to scrape 1250 pages as this was over half of all of the pages and I did not want to risk getting banned from StackOverflow (I live in the dorms so I am on the University’s network) and be unable to do the assignment. This gave me a dataframe with 62500 observations and 10 variables. This is a large segment of the population and should allow my responses to be fairly accurate.

StackDataset = StackOverflow(pagelimit = 1250)

1. For Part 3 we begin by taking the rQAs dataset that was provided and subsetting it to only get those who answered a question, then taking a table of these values and putting them into a histogram.

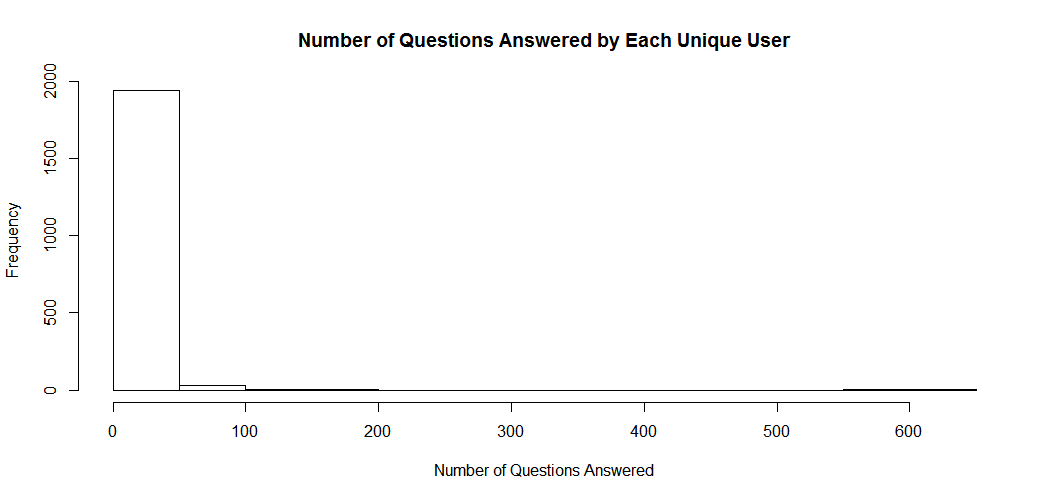
answersub = subset(rQAs, rQAs$type == "answer")

answertable = table(answersub$user)

hist(answertable, xlab = "Number of Questions Answered",

main = "Number of Questions Answered by Each Unique User")

We get this histogram:

  
This shows that most users answer very few questions with a small minority answering over 500 questions each with the maximum being 642 answers to questions. This leads to a vastly skewed right response on the histogram.

2. For this problem we go to the data that I scraped in part 1. I took the column called “tags” in this one and used strsplit in an sapply function to split up the pasted together tags for each line.

taglist = sapply(1: length(StackDataset$Tags), function(i) strsplit(StackDataset$Tags[i], "; "))

taglist = unlist(taglist)

tagtable = table(taglist)

head(sort(tagtable, decreasing = TRUE))

I then unlisted them after the saply and looked at the top 6 most occurring values.

r ggplot2 plot data.frame shiny data.table

62500 5062 2825 2798 2588 1953

The r tag makes sense due to the data being scraped from the r tag itself, ggplot2 is interesting but also makes sense due to it being a powerful package with lots of abilities, obviously plotting is still an issue with plot being the 3rd most common tag. Dataframes and tables are also important with it being at 4 and 6. Shiny, which is some other R tool that I have never used is interesting coming in at number 5.

3. For finding the number of questions that were regarding ggplot, I went back to table that I created in question 2 to search for any tags containing ggplot or ggplot2, I will also include qplot since that is a commonly used function from the ggplot2 package. When looking through the number of tags that had ggplot2 I realized that there are no tags made up for just ggplot or qplot that exist. Therefore using the following argument: table(taglist[taglist %in% "ggplot2"]) we get the following response



This is 8.1% of all of the tags I received that were under ggplot2

4. For this problem I had to take a somewhat different direction but similar to the third question. Since looking at xml, html, and Web Scraping (in this case it was under the description of web-scraping) and that they are not independent of one another if we just add up all the times they occurred together. I had to make a new function that was a for loop nested inside another for loop where it would check for each of the tags and if one of them existed it would add to a count and then break out of the current loop to go to the next value. Running this function to check for everything gave me back **842** questions that had any of the hits of these.

taglistlist = sapply(1: length(StackDataset$Tags), function(i) strsplit(StackDataset$Tags[i], "; "))

Q4function = function(hugelist){

questioncount = 0

for(i in 1:length(hugelist)) {

for(w in 1:length(hugelist[[i]])) {

if(hugelist[[i]][w] == "xml") {

questioncount = questioncount + 1

break

}

else if (hugelist[[i]][w] == "html"){

questioncount = questioncount + 1

break

}

else if (hugelist[[i]][w] == "web-scraping"){

questioncount = questioncount + 1

break

} } }

return(questioncount)

}

Q4function(taglistlist)

5. To start off this question I found a function online that gets a list of functions from each package. Using this I found a list of functions from the base r package along with ggplot2 and plyr since these are very popular packages. To do this I first subset rQAs by making type == ‘question’ and then made a function that took in the text and ran a grepl function over each value and then adding 1 to the count of the function if it exists.  
Q5function = function(tablelist, datasubset)  
{

for(i in 1:length(datasubset$text)) {

textvalue = datasubset$text[i]

for(x in 1:length(tablelist)) {

greptest = grepl(names(tablelist[x]), textvalue, ignore.case = TRUE)

if(greptest == TRUE) {

tablelist[[x]] = tablelist[[x]] + 1

} } }

return(tablelist)

}

Looking at the data, we see that the most popular functions tend to be just letters (c and t appear in all values) and since the title of rQAs is in html, class is in all titles. Because of this here are the top 54 functions that showed up.



As we can see the functions that were used that are more than likely used as functions rather than just as actual words we see that there is data.frame, ggplot, min, exp, and apply as some of the major ones.

6. For this problem we were to look at the accepted comments and answers for posts. With this restriction, I subsetted based upon the type being answer or comment and then subsetted again on if the score was either “” (common for the comments) or if they were greater than 0 (a comment would, I would guess, be accepted if at least one person upvoted it)

thetopones = subset(rQAs, rQAs$type == c("answer", "comment"))

thetopones = subset(thetopones, thetopones$score == "" | thetopones$score > 0)  
Q6ans = Q5function(tablefunction, thetopones)

head(sort(Q6ans, decreasing = TRUE), n = 54)

I then ran the function from the previous question. This had the same issues as before with the most popular ones being letters or common words. Ones that made it through that were functions included data.frame again, apply, min, and textConnection when looking at the top 54 values. 