STA141 Assignment 6

Chad Pickering

Wednesday, December 09, 2015

Non-copying statement: I did this assignment by myself and developed and wrote the code for each part by myself, drawing only from class, section, Piazza posts and the Web. I did not use code from a fellow student or a tutor or any other individual.

Signed:

Resources: Office hours notes, Discussion notes, Lecture example on Cybercoders, w3schools XML tutorial, StackOverflow (both for scraping and for the function writing process, especially the "next button" part)

Preface: In this entire assignment, due to some of the vague phrasing in Parts 1 and 3, I set up my large function in a way that makes sense to me - without the smaller functions (analysis below) - and I was able to alternate between using the final data frame I created in Part 1 and the given data frame in Part 3 for the 6 exploratory questions. My determinations on which to use are detailed below, but the main overarching reasoning was because some information (such as tags) were not provided in the given data frame, so I had to work with what I had extracted. I hope these interpretations of the directions for both required parts of this assignment are acceptable.

```r
load("C:/Users/cpickering/Syncplicity Folders/ChadSync/STATISTICS/STA141/rQAs.rda")
```

I found a very peculiar attribute in the data frame given to us in Part 3. The first column of data looks like any other, with a column name "row.names" and content that did not look like expected "primary key" like information, and yet the column's name did not show up when I called names() on the data frame (and thus I could not extract that column for manipulation). I had to cbind() this column (the rownames it turns out) to the other side of the data frame and then assign a different name to it in order to use it with convenience in Part 3. I suspect that when this data frame was created the option to make the first column the key was selected. I was pleased to see a use of this in action since I have manipulated those settings before on other data sets but have never chosen that option.

```r
rQAs <- cbind(rQAs, rownames(rQAs))
colnames(rQAs)[11] <- "title"
```

Part One: Web Scraping Post Summaries I heeded the advice of making two arguments in my main function - the language or topic (tag) that wants to be searched (and therefore pasted between the constant portions of the URL), and the number of pages wished to be scraped. This value has a default, set to "max", or the very last page of the certain tag, so if no value is specified when the function is called, then all pages will be scraped. I originally had a small function for each of the columns created (most with only one argument - html) that were called before the data frame creation, but I found them to be rather pointless beyond debugging purposes in each section. The html is constant across all iterations, so I found it illogical to have lots of small functions just for the purpose of having functions that satisfied the endorsed methods for this assignment. These functions also made the entire process harder to understand if someone were to look at the code with (or without) comments, in my opinion.
When the user inputs a tag/language/topic, the function first has to assemble the URL. Using the head and tail pieces constructed outside the main function, the base URL is created by pasting the tag inputted in the function between them. This URL is used to then find the second to last link at the bottom of the page that contains the maximum page value at the current time. This is set to what I call "max", the limit to how many pages can be scraped; if the user inputs a value over this amount, the for loop just breaks when the value that is being iterated reaches one more than the maximum. So, whatever value over the maximum the user inputs, the for loop will simply iterate until the maximum value is reached. I used this for loop break

method in Java, and Nick's notes from office hours and discussion helped solidify these concepts. Looking at the "while" loop example from lecture also helped in terms of what is possible for this situation. We had not emphasized many C/C++ like syntax so far in this class, so it was nice to revisit some.
To get the "next page" link in order to iterate, I used a very similar method to what I did to get the maximum page value. However, instead of using regular expressions to grab a value within the URL, I just grabbed the appropriate URL segment itself and used getRelativeURL() to complete it and store it back into the variable that is used at the top of the for loop. This process of scraping an individual page then grabbing the URL for the next page continues until the user-inputted limit is reached.
As I tested this function more and more, I became aware of users that do not have accounts, so where other users have a link, these users have a completely different XPath. Because each part of the required analysis does not depend on relationships between the variables for any particular row(s), I did not find it pertinent to fix this issue. In part three, we are concerned more about relationships **within** variables, so when I had to use my data frame from part one, any discrepancies between rows have no bearing on within- column accuracy.
I have commented the entirety of Part 1 to demonstrate my understanding along the way - please see these.

```r
library(XML)
library(RCurl)
```

```
## Loading required package: bitops
```

```r
# Establishes constant URL sections before and after the tag variable
head_url <- "http://stackoverflow.com/questions/tagged/"
tail_url <- "?sort=newest&pagesize=50"

# Complete function - input language/topic tag, and page limit (by default maximum)
scrape_stacko <- function( lang_tag, page = max )
{
  # Right away, create empty data frame so that it can be filled later
  df_all <- data.frame()

  ### Process to create base URL:

  # Pastes the constant bookends and the user-inputted tag/topic - this has unwanted spaces
  paste_url <- paste(head_url, lang_tag, tail_url)

  # Use gsub() to collapse the spaces - this is the base URL to work with
  use_url <- gsub(" ", "", paste_url)

  # Get content from StackOverflow [to find max page below]
  doc <- getURLContent(use_url)

  # Organize content in output
  html <- htmlParse(doc, asText = TRUE)

  ### Process to find maximum page:

  # XPath code that references the links at the bottom of the StackOverflow page
  nexturl_path <- "//div[@class = 'pager fl']/a/@href"

  # Gets list of six tail URL segments for the links at the bottom of each page -
  # (Plus an irrevelant seventh element)
```

```r
raw_nextlinks <- getNodeSet(html, nexturl_path)

# Unlists the links/elements
all_nextlinks <- unlist(raw_nextlinks)

# Since length of list is 6, the fifth element is needed -
# the button with the maximum page value
last_nextlink <- length(all_nextlinks) - 1

# Gets the fifth element, as discussed
max_pagevals <- all_nextlinks[last_nextlink]

# In the URL to get the max (last) page, the actual value must be extracted -
# Use reg. expression (will always be a unique four digit value)
max_pagenum <- regexec("[0-9]{4}", max_pagevals)

# Use regmatches() to output result of reg. expression; convert character to numeric
max <- as.numeric(regmatches(max_pagevals, max_pagenum))

# For loop to iterate from page 1 until the page limit inputted up top in main function
for( x in 1:page )
{
  # If the inputted page limit is greater than the maximum calculated above, break
  # Inspired by Duncan's while loop example
  if( x > max )
    break

  # Get content in organized form [need this in each iteration of for loop]
  doc <- getURLContent(use_url)
  html <- htmlParse(doc, asText = TRUE)

  ### Actual questions asked in part one:

  ## Who posted it:
  who_path <- "//div[@class = 'user-details']/a/text()"
  who_col <- xpathSApply(html, who_path, xmlValue)

  ## When it was posted:
  when_path <- "//div[@class = 'user-action-time']/span/@title"
  when_col <- unlist(getNodeSet(html, when_path))
  names(when_col) <- NULL

  ## Title of the post:
  title_path <- "//div[@class = 'summary']/h3/a/text()"
  title_col <- xpathSApply(html, title_path, xmlValue)

  ## The reputation level of the poster:
  rep_path <- "//div[@class = 'user-details']/span/text()"
  rep_col <- xpathSApply(html, rep_path, xmlValue)
  # Under 20000, k means '00'; Above 20000, k means '000'
  rep_col <- gsub(",", "", rep_col)

  ## The current number of views:
  views_path <- "//div[@class = 'views ']/text()"
  views_raw <- xpathSApply(html, views_path, xmlValue)
  views_col <- gsub("\r|\n|views?|,|[[:space:]]|", "", views_raw)

  ## The current number of answers:
  answers_path <- "//div[@class = 'status unanswered' or @class = 'status answered' or @class = 'statu
  answers_col <- xpathSApply(html, answers_path, xmlValue)

  ## The vote "score" for the post:
  vote_path <- "//span[@class = 'vote-count-post ']/strong/text()"
  vote_col <- xpathSApply(html, vote_path, xmlValue)

  ## URL for the post itself:
  postURL_path <- "//div[@class = 'summary']/h3/a/@href"
  post_urls <- xpathSApply(html, postURL_path, '[[', 1)
  posturl_col <- getRelativeURL(post_urls, use_url)
  names(posturl_col) <- NULL

  ## The ID/number uniquely identifying the post:
  numid_path <- "//div[@class = 'summary']/h3/a/@href"
  idloc_raw <- xpathSApply(html, numid_path, '[[', 1)
  idloc_ind <- regexec("\\/questions/([0-9]+)\\/", idloc_raw)
  idloc_match <- regmatches(idloc_raw, idloc_ind)
  id_col <- sapply(idloc_match, function(x) x[2])

  ## The tags of the post:
  tag_path <- "//div[starts-with(@class,'tags')]"
  tag_raw <- xpathSApply(html, tag_path, xmlValue, trim = TRUE)
  tag_col <- gsub(" ", ";", tag_raw)

  ## Individual tags - not used, but relevant:
  tag_test <- "//div[contains(@class,'tags')]/a/text()"
  tag_raw_col <- xpathSApply(html, tag_test, xmlValue, trim = TRUE)

  ### Data frame creation:

  # cbind() columns created above into data frame from the i'th iteration
  df_additional <- cbind(id_col, who_col, when_col, title_col, rep_col,
                     views_col, answers_col, vote_col, posturl_col, tag_col)

  # rbind() the additional 50-row sub-data frame to what already exists (df_all)
  df_all <- rbind(df_all, df_additional)

  ### Iterate to the next page:

  # Just like outside of the for loop above, need to access the buttons along the bottom again -
  # this time to get the "next" button to proceed to the next page (in the next iteration)
  nextlinks_raw <- getNodeSet(html, nexturl_path)
  nextlinks_all <- unlist(nextlinks_raw)

  # Last time I wanted the next to last button to extract the maximum page value -
  # this time I want the last value (6), which truly links to the next page
  nextlinks_len <- length(nextlinks_all)
```

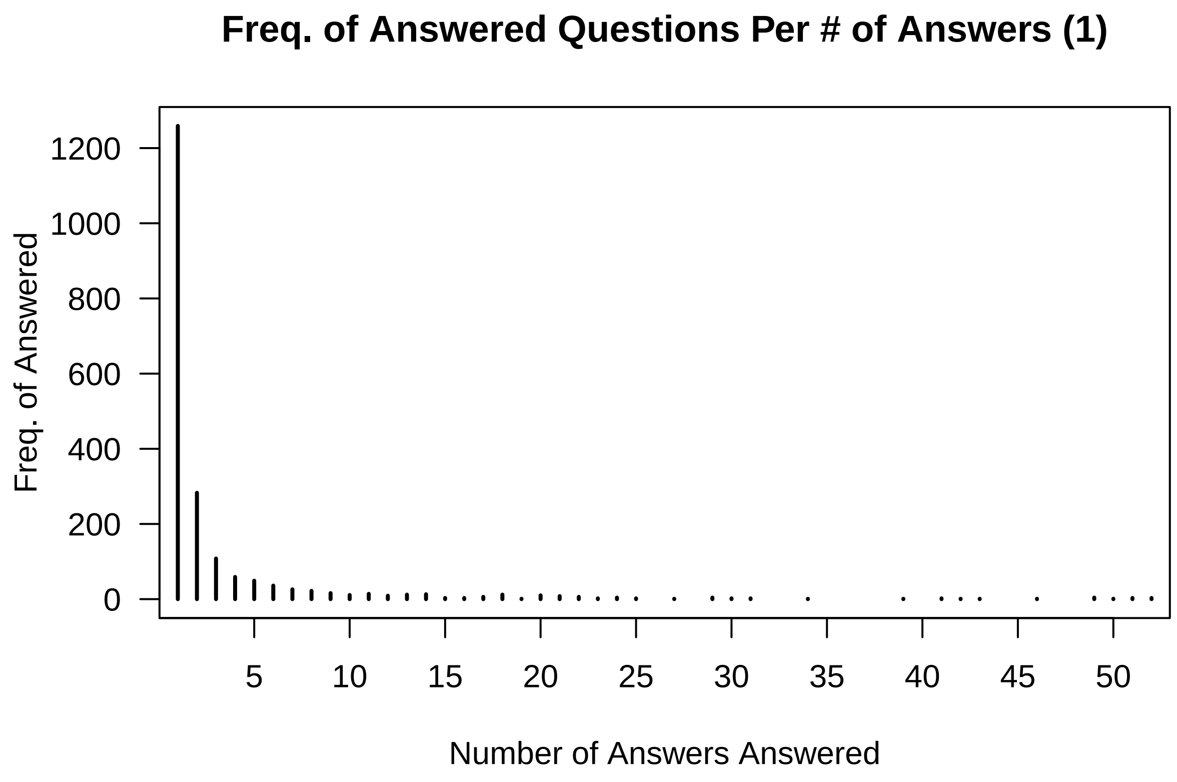
```r
    # Gets the actual last URL (sixth element)
    nextlinks_sub <- nextlinks_all[nextlinks_len]

    # Use getRelativeURL() to create complete URL of "next page" link
    nextlinks_rel <- getRelativeURL(nextlinks_sub, use_url)

    # Store in the use_url variable that gets called with getURLContent() at the top of the for loop
    use_url <- nextlinks_rel
  }

  # return final data frame with all posts in all pages scraped
  return(df_all)

}

# Scrape 150 pages in the "r" tag (good "sample" size of 7500 posts)
df_final <- scrape_stacko('r', 150)

# Example of data frame output
head(df_final, 5)
```

```
##     id_col    who_col             when_col
## 1 34145865 Ryan Erwin 2015-12-08 00:19:51Z
## 2 34145794       user 2015-12-08 00:12:44Z
## 3 34145787  tssssddde 2015-12-08 00:12:01Z
## 4 34145665      shadi 2015-12-08 00:00:32Z
## 5 34145522      Mohaa 2015-12-07 23:47:11Z
##                                                                     title_col
## 1                                         Rmarkdown Image Skips Ahead of Text
## 2                                     Edit a record in a datatable in shiny r
## 3                           Best way to count unique element in a string in r
## 4                                                    Reordering the variables
## 5 Check the visibility of an object in a webpage using its xpath? [duplicate]
##   views_col answers_col vote_col
## 1         4           0        0
## 2         4           0        0
## 3         7           0        0
## 4        11           1        0
## 5         6           0        0
##                                                                                                 pos
## 1                            http://stackoverflow.com/questions/34145865/rmarkdown-image-skips-ahead-
## 2                        http://stackoverflow.com/questions/34145794/edit-a-record-in-a-datatable-in-
## 3              http://stackoverflow.com/questions/34145787/best-way-to-count-unique-element-in-a-stri
## 4                                       http://stackoverflow.com/questions/34145665/reordering-the-va
## 5 http://stackoverflow.com/questions/34145522/check-the-visibility-of-an-object-in-a-webpage-using-i
##                             tag_col
## 1                 r;knitr;rmarkdown
## 2         r;shiny;shinydashboard;dt
## 3                          r;unique
## 4                         r;program
## 5 r;selenium;web-scraping;rselenium
```

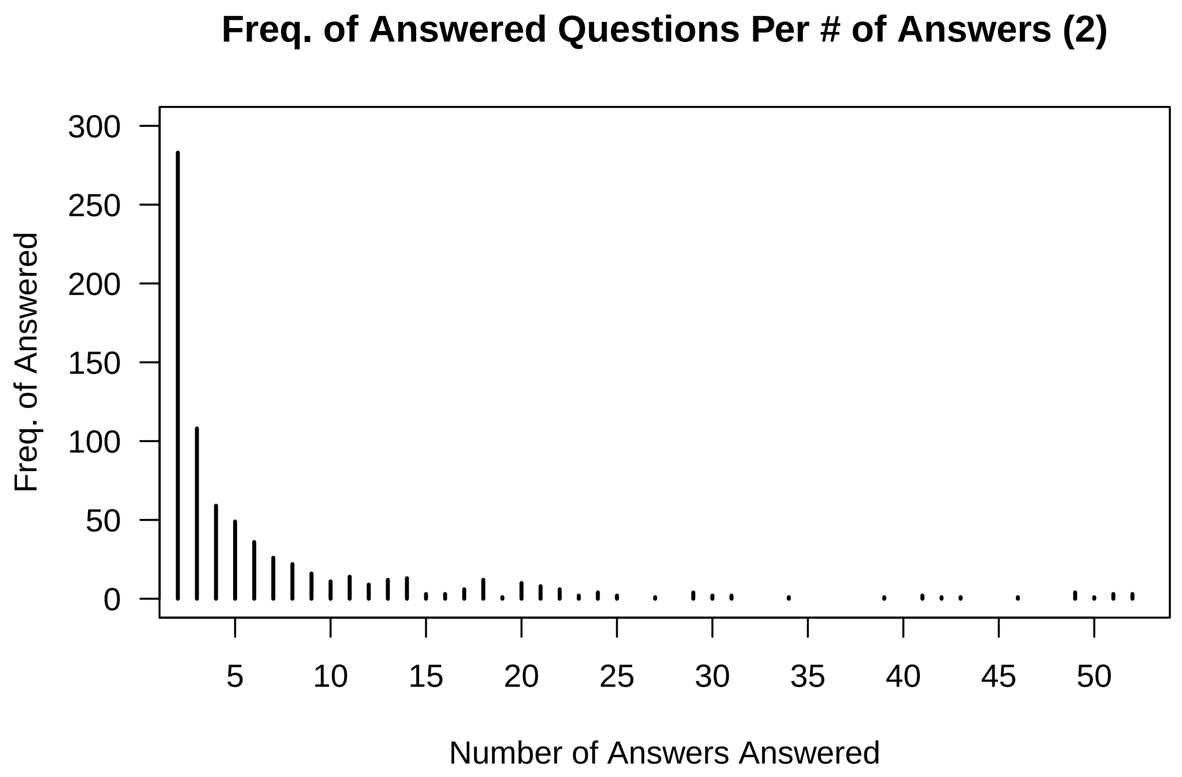
Part Three: Specific R Questions I will be very specific along the way in these six follow-up questions as to what data I am using. Some questions are only possible to answer when using the data scraped from part 1.

Question 1: What is the distribution of the number of questions each person answered? Using the rQAs data frame, it becomes clear that the number of questions users answer is greatly concentrated near zero. A vast majority of users only answer less than around 5 questions, while only a small handful answer dozens, if not hundreds of questions (at least within the scope of the provided data frame, rQAs). This demonstrates that there are only a handful of very active users that have experience in programming and are either employed by the company or are educators elsewhere.
I determined that the distribution is exponential decay with a very high lambda/k value (the y intercept). After I plotted the first graph with limits between 0 and 50, I wanted to demonstrate the sort of "memoryless"-esque property of exponential decay that these plots mimic, by restricting the x limits to just beyond 1 and just beyond that again. In both of these plots, if you treat the lower x limit as the origin (zero), the exponential decay is still obvious.
There are not many other embellishments I can make to these plots, so demonstrating the property was appropriate.

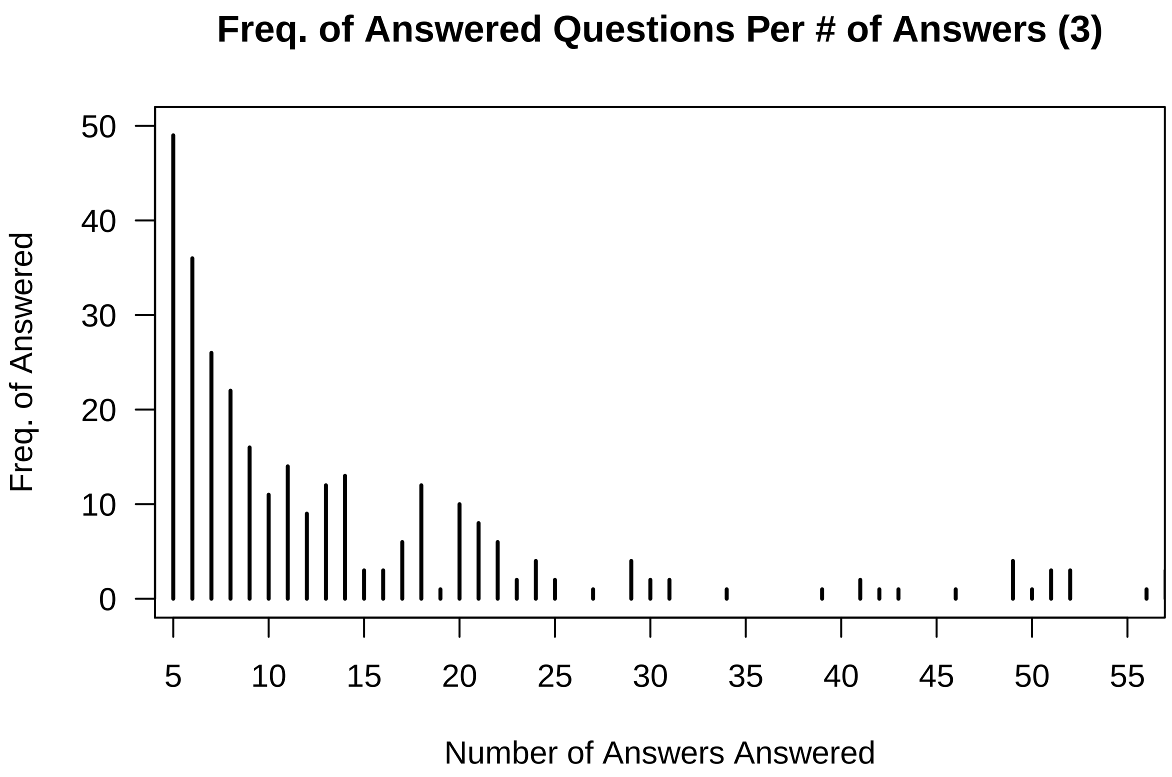
```r
all_answers <- rQAs[rQAs$type == "answer",]
# as.factor used to make only the counts with >0 frequency actual levels
uniqueid_freq <- as.factor(sort(table(all_answers$userid), decreasing = TRUE))

# 1 to 50 responses frequencies
plot(table(uniqueid_freq), main = "Freq. of Answered Questions Per # of Answers (1)",
     xlab = "Number of Answers Answered", ylab = "Freq. of Answered",
     xlim = c(2,51), xaxt = "n", las = 2)
axis(1, xaxp = c(0, 50, 10), las = 1)
```



```r
# 2 to just beyond 50 to demonstrate that the exponential distribution continues to matter
# where you start counting / use as the origin
plot(table(uniqueid_freq), main = "Freq. of Answered Questions Per # of Answers (2)",
     xlab = "Number of Answers Answered", ylab = "Freq. of Answered",
     xlim = c(3,52), ylim = c(0,300), xaxt = "n", las = 2)
axis(1, xaxp = c(0, 50, 10), las = 1)
```



```
# 5 to beyond 55 to continue demonstration of previous concept; beyond 25 is 0 to 5 frequency
# per unique number of answers
plot(table(uniqueid_freq), main = "Freq. of Answered Questions Per # of Answers (3)",
     xlab = "Number of Answers Answered", ylab = "Freq. of Answered",
     xlim = c(6,55), ylim = c(0,50), xaxt = "n", las = 2)
axis(1, xaxp = c(5, 55, 10), las = 1)
```



Question 2: What are the most common tags? For this question, I had to use the data that I scraped in Part 1. If you notice, I included another bullet point in Part 1 so that I scrape all tags for each post. I did this in two ways: I both extracted the tags for each post in particular and separated each tag by a semi-colon, and I extracted all individual tags unconditional on post, so that I could find most common frequencies of particular tags and groupings of tags.
The package ggplot2 for graphics is the most frequent non-"r" tag used, meaning that most questions have to do with graphical formatting and syntax issues. This would suggest that the documentation for ggplot2 needs to be either improved drastically or needs to be more easily accessible to those who need it. This is also true for the data frame documentation and Shiny, the framework for interactive web applications in R - these are also high on the list in terms of individual tags. For groups of tags, since "r" is the constant tag based on the input into the Part 1 function, it is apparently common to only put one other tag on your post; this second tag describes the root of the question the user has.
Beyond this inerpretation, I can certainly see this kind of web scraping of tags useful in many more ways than just here. Like I was mentioning, based on these common sources of problems in R that users have, the number of tags can tell R architects what packages and/or documentation to improve and have readily available.

```r
# Tags in a full string per post
head(sort(table(df_final$tag_col), decreasing = TRUE), 20)
```

```
##
##             r      r;ggplot2   r;data.table        r;dplyr        r;shiny
##          1367            238            133            120             92
##  r;data.frame         r;plot       r;matrix        regex;r      r;rstudio
##            71             60             57             45             32
##          r;csv r;plot;ggplot2   r;statistics        r;loops      r;r-caret
##             30             23             21             21             20
##         r;date       r;igraph       r;subset   r;regression        r;knitr
##             19             19             19             18             18
```

```
# Individual tags
ind_tags <- strsplit(as.character(df_final$tag_col), ";")
head(sort(table(unlist(ind_tags)), decreasing = TRUE), 20)
```

```
## 
##          r    ggplot2      shiny       plot data.frame data.table 
##       7500        625        323        315        265        264 
##      dplyr     matrix    rstudio      regex      loops      knitr 
##        260        171        162        141        122        112 
##   function statistics        csv  rmarkdown   for-loop       list 
##        107         99         97         97         96         96 
##     string     subset 
##         93         86
```

Question 3. How many questions are about ggplot? For this question, I used the data frame provided in Part 3, rQAs. I subsetted the data frame so that I just regard questions. Then, in both the text and title columns, separately, I searched for the term "ggplot" - this gave me 959 matches in the text and 516 in the title. Both of these values would be variable if I were using my Part 1 data frame so this would update/change when more questions are posted AND when more or less pages are scraped. Here, though, with rQAs, these values are constant.
Then, I wanted to find out what proportion of post that have the tag "ggplot" or "ggplot2" have the term "ggplot" in the title. Using grep() and subsetting on the Part 1 data frame, I was able to determine that about 65% (variable) of "ggplot" tagged posts have the same term in the title. This investigation interested me because I wanted to see if merely looking at a post's title can give most or all insight as to what the apparent root of the problem/question is. Most of the time, this is true, but it seems as if we also must look at the body of the question to determine the most frequent terms/functions/keywords to be sure.

```
all_questions <- rQAs[rQAs$type == "question",]

# 959 posts mention ggplot in the question's text.
ggplot_text <- grep("ggplot", all_questions$text, ignore.case = TRUE)
length(ggplot_text)
```

```
## [1] 959
```

```
# 516 posts mention ggplot in the question's title.
ggplot_title <- grep("ggplot", all_questions$title, ignore.case = TRUE)
length(ggplot_title)
```

```
## [1] 516
```

```
# ~65% of the posts that have the tag 'ggplot' have the term 'ggplot' in the title.
ggplot_tags <- grep("ggplot", df_final$tag_col, ignore.case = TRUE)
ggplottag_titles <- df_final[ggplot_tags, 'title_col']
ggplot_titletag <- grep("ggplot", ggplottag_titles, ignore.case = TRUE)
titlebytag_3 <- length(ggplot_titletag)/length(ggplottag_titles)
titlebytag_3
```

```
## [1] 0.64
```

Question 4. How many questions involve XML, HTML or Web Scraping? For this question, I used the data frame provided in Part 3, rQAs. This process is the same as in question 3, and I got 1224 questions that mention the three terms in the text, and 138 questions that mention any of the three terms in the title. I continued my extra investigation as well, to see if the trend continued, and with my result here of around 60% (variable), I can say that, while I would have to test several dozen or even hundred more terms/tags, this trend seems to have some sort of validity. I would expect a majority of posts that have tag X to have that specific term mentioned verbatum in the title. To reach a conclusion with support, I would need to repeat this process on a random sample of tags.

```r
# 1224 questions mention XML, HTML, or web scraping in the text.
scrape_text <- grep("xml|html|web scraping", all_questions$text, ignore.case = TRUE)
length(scrape_text)
```

```
## [1] 1224
```

```r
# 138 questions mention XML, HTML, or web scraping in the title.
scrape_title <- grep("xml|html|web scraping", all_questions$title, ignore.case = TRUE)
length(scrape_title)
```

```
## [1] 138
```

```r
# ~60% of the posts that have the tags 'xml', 'html', or 'web-scraping' have those terms in the title.
scrape_tags <- grep("xml|html|web-scraping", df_final$tag_col, ignore.case = TRUE)
scrapetags_titles <- df_final[scrape_tags, 'title_col']
scrape_titletags <- grep("xml|html|web-scraping", scrapetags_titles, ignore.case = TRUE)
titlebytag_4 <- length(scrape_titletags)/length(scrapetags_titles)
titlebytag_4
```

```
## [1] 0.5819672
```

Question 5. What are the names of the R functions referenced in the titles of the posts? For this question, I used the Part 1 data frame instead of the rQAs (given) data frame because the titles of the posts are in the form of URLs; I wanted to use legitimate titles. Here, I used a regular expression to match all letters, an optional period, and an optional open parentheses. I then compared this output to a list of legitimate R functions (Piazza @1568); I matched any in my list that corresponded to any member in the comprehensive list and found 222 total unique R functions referenced in the titles of the posts. Some of the functions I matched look very bizarre, but I trust the comprehensive built-in list of base R functions, so I will not try to filter further at risk of removing legitimate functions.

```r
func_regex <- gregexpr("[a-z]+\\.?\\(?", df_final$title, ignore.case = TRUE)
regex_match <- regmatches(df_final$title, func_regex)
all_ext_func <- gsub("\\(", "", unlist(regex_match))
uq_func <- unique(all_ext_func)

# list of all R functions - found on Piazza @1568
functions_list <- ls(getNamespace("base"), all.names = TRUE)

# gives all indices where the extracted function matches with a legitimate function from list
realfunc_idx <- which(uq_func %in% functions_list)
legit_func <- uq_func[realfunc_idx]

# ~220 (variable) legitimate functions found
length(legit_func)
```

```
## [1] 221
```

```
head(legit_func, 30)
```

```
##  [1] "unique"      "function"    "for"         "interaction" "return"
##  [6] "labels"      "Find"        "within"      "t"           "assign"
## [11] "row"         "with"        "if"          "exists"      "table"
## [16] "I"           "list"        "names"       "cut"         "file"
## [21] "character"   "integer"     "vector"      "numeric"     "match"
## [26] "factorial"   "solve"       "version"     "ordered"     "by"
```

Question 6. What are the names of the R functions referenced in the accepted answers and comments of the posts? Finally, here, I used the Part 3 given data and followed the same procedure as in question 5. I followed the instructions and subsetted correctly and used the regular expression on the text of the posts. In this go through, I noticed curiosities in the results. Even when comparing with the legitimate functions, I got quite a few in my final list that are obviously not functions; however, I stand by what I said in question 5 - I certainly don't know most exotic function names, so anything is possible.
For both questions 5 and 6, these strategies can be used to identify keywords, tags, etc. for the investigation I introduced in questions 3 and 4.

```
all_anscom <- rQAs[rQAs$type == "answer" | rQAs$type == "comment",]

func_regex6 <- gregexpr("[a-z]+\\.?\\(?", all_anscom$text, ignore.case = TRUE)
regex_match6 <- regmatches(all_anscom$text, func_regex6)
all_ext_func6 <- gsub("\\(", "", unlist(regex_match6))
uq_func6 <- unique(all_ext_func6)

# gives all indices where the extracted function matches with a legitimate function from list
realfunc_idx6 <- which(uq_func6 %in% functions_list)
legit_func6 <- uq_func6[realfunc_idx6]

# 394 "legitimate" functions found
length(na.omit(legit_func6))
```

```
## [1] 394
```

```
head(legit_func6, 30)
```

```
##  [1] "class"     "comment"   "list"      "character" "unlist"
##  [6] "factor"    "length"    "which"     "I"         "function"
## [11] "with"      "by"        "solve"     "for"       "library"
## [16] "replicate" "cbind"     "try"       "scale"     "if"
## [21] "t"         "structure" "range"     "max"       "abs"
## [26] "rep"       "c"         "subset"    "labels"    "unique"
```