Katherine Olson

STA 141 Assignment 6

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.

Report

Part I: Scraping the Summaries of the Posts

I ended up scraping 100 pages, resulting in a data frame with 5000 rows.

Title: xpath = "//div[@class = 'summary']/h3/a"

> This xpath was one of the less challenging ones, I grabbed the title from the summary class. The con of this is that if the class changes or the title gets moved from h3, my code will no longer work. The pro is how that would be one small fix, but it could take a while to pin point the problem. I just looked at the titles for the first however many rows of the resulting data frame, and they all looked correct.

Date: xpath = "//div[@class = 'user-action-time']/span"

> This xpath was a little tricky since it was my first time dealing with the xmlGetAttr(), function. I called that function with "title" to get my answer. I also added a check to deal with NAs. This xpath has similar pros and cons to the previous one, the website can change where the date is stored, but rearranging the way stackoverflow is formatted could upset some regular users. Looking at the dates from the first page, they all look correct.

Reputation: xpath = "//div[@class = 'user-details']"

> I struggled with this one for a while, after having it fail on page 7 with a user that had no reputation score and therefore the class I was using before did not exist for that user. So I had to start with the user-detail class, loop over each one, and use xmlChildren to move my way down to the correct node. I do not think there is a simpler way to do this, especially for scraping so many posts where there is bound to be someone who is missing a reputation score. The pro of this is that the class user-details will probably not be changed, but the location of the reputation score could be changed. Looking at the results from the first page, there were some values with commas in the thousands place, so I had to leave the results as character strings. This could be changed if necessary.

Views: xpath = "//div[@class = 'views ']"

The tricky part about this one is the space after the s in the views class. Someone might notice this mistake and remove the space. This one was also a little tricky with how the result had to be manipulated to pull out the view from the string. The results I got looked correct and I got no errors or warnings with the use of as.numeric(), so that was a good sign.

Answers: xpath = "//div[@class = 'status answered' or @class = 'status unanswered' or @class = 'status answered-accepted'

The hard part about this one was that I just had the first two conditions and the third condition popped up when I got to page two and discovered a third class relating to the number of answers. This is dangerous because the classes could be changed, maybe there is even a status answered-rejected class that I did not come across with the first 100 pages. My results looked correct.

Votes: xpath = "//div[@class = 'votes']"

I followed a similar process for dealing with the way the votes were stored as I did with views. Things could always change, but this xpath seems fairly stable and unlikely to be changed. My results looked correct for the first page.

Url: "//div[@class = 'summary']/h3/a"

The trickiest part about this one was finding the url and making sure it was the right one. Then adding on the front part of the url made it a little trickier. The url could be moved somewhere else, but that is the risk we take. My results looked correct.

ID: "//div[@class = 'question-summary']"

Finding the ID and using substring to get the ID part made this one a little challenging. I saw that some people on piazza were saying that they got the ID from the url, which is another way of finding it. They both have risks of being changed. My results looked correct

Poster: xpath = "//div[@class = 'user-details']"

This one got a little tricky with needing strsplit() and needing to watch out for NAs, but otherwise was not too bad. This one has the same pros and cons as usual and my results for the first 50 looked correct and were the same posters I saw when I looked at the page myself.

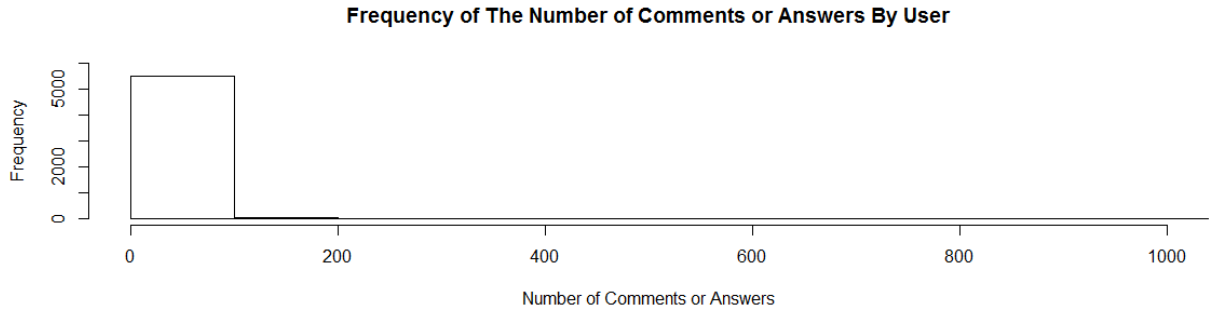
Tags: xpath = "//div[contains(@class, 'tags ')]"

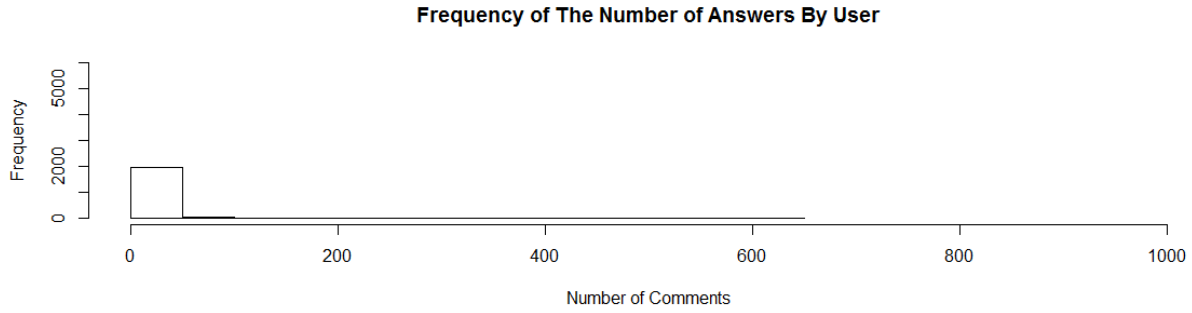
The most challenging part about this one was figured out that the tags were stored with a strange t- as the start and figuring out how to extract the tags. It was nice to know that all of them had tags since we were scraping data with r as a tag. Since the way the tags were stored was so strange, at least to me, stackoverflow could feel the same way and completely change the way they are stored. My results looked correct.

Part III: Analyzing R Questions on StackOverflow

Question 1: What is the distribution of the number of questions each person answered?

I separated the plots out into posts that were comments and answers and posts that were just answers. Both are left skewed with a few high question answers that can barely be seen. So most people that comment or answer a post have only made less than 100 other posts, and there are a few crazy but helpful other people.





Question 2: What are the most common tags?

Using the 5000 posts I scraped from part I, I got the results below for the top 10 tags:

r	ggplot2	plot	shiny	dataÃƒÂ»frame	dplyr	dataÃƒÂ»table	matrix	rstudio	regex
5000	405	210	198	188	184	171	124	106	91

It makes sense that r is the top tag with 5000 results since we scraped questions with the tag r. The rest all make sense with tags on plotting, the way data is stored, regex, and so on. The only one that seems weird to me is shiny. After using the web, it is clear that shiny is a package, so is does not seem so weird anymore.

Question 3: How many questions are about ggplot?

I interpreted this question as tagged ggplot because of the use of "about." Looking at the table from question two, I found 405 questions about ggplot2 in the 5000 posts I scraped.

Question 4: How many questions involve XML, HTML or Web Scraping?

I interpreted this question because of "involve" as questions that have the above terms in the text. I used the rQAs data for this question, so the results cannot be compared to the results from question 3. I used the regular expression "xml|html|web scraping" and found 1327 questions.

Question 5: What are the names of the R functions referenced in the titles of the posts?

For this question I used the 5000 posts I scraped and the regular expression "[a-z][a-zA-z]{3,19}" to search through the titles. After applying unique, I ended up with 5054 words and got stuck. I considered comparing the words to the base functions, but that would leave a lot out. The first 6 of the 5054 are below. It is clear that a lot of these are not functions.

"markdown" "table" "pushed" "pages" "when" "overcome"

Question 6: What are the names of the R functions referenced in the accepted answers and comments of the posts?

For this question I used the rQAs data and the regular expression "[a-z][a-zA-z]{3,19}[(]" on a subset of the data including only answers and comments. I searched through the text column and was able to add a (at the end of my expression and ended up with 4693 results. The first 6 can be seen below. There are probably some results that are not functions, but much less than for question 5.

"factor(" "unlist(" "mutate(" "list(" "untar(" "expand("

Resources

- Piazza
- Class website
- Lecture and discussion
- http://stackoverflow.com/questions/7723549/getting-and-removing-the-first-character-of-a-string
- http://www.inside-r.org/r-doc/base/sprintf

Code Appendix

```
# Part I Scraping the Summaries of the Posts
# ================================
library("RCurl")
```

```
install.packages("XML")
library("XML")

res =  AllPages("r", 100)

AllPages = function(tag, numPages){

  mainURL =
sprintf("http://stackoverflow.com/questions/tagged/%s?sort=newest&pagesize=50", tag)
  mainURL = "http://stackoverflow.com/questions/tagged/r?page=7&sort=newest&pagesize=50"
  firstPage = getURLContent(mainURL, verbose = TRUE)
  doc = htmlParse(firstPage, asText = TRUE)

  scrape = OnePage(doc)
  doc = GetNextDoc(doc)

  while(nrow(scrape) < (numPages * 50)){
    # Get dataframe for the page:
    page = OnePage(doc)
    # Add it onto entire dataframe:
    scrape = rbind(scrape, page)
    saveRDS(scrape, "scrape.rds")
    doc = GetNextDoc(doc)

  }
  return(scrape)
}

# Function that takes in doc(one page that was htmlParsed), find the url for the next page,
# htmlparses that page, saves it, and returns it:
GetNextDoc = function(doc){
  nextURL = getNodeSet(doc, "//div/a[@rel = 'next']")
  nextURL = xmlGetAttr(nextURL[[1]], "href")
  nextURL = sprintf("http://stackoverflow.com%s", nextURL)
  txt = try(getURLContent(nextURL, verbose = TRUE))
  nextDoc = htmlParse(txt, asText = TRUE)
  return(nextDoc)
}

# Function takes in doc (a page that was htmlParsed) and returns a vector of titles:
```

```
Title = function(doc){
  titles = getNodeSet(doc, "//div[@class = 'summary']/h3/a")
  titles = sapply(titles, xmlValue, trim = TRUE)
  return(titles)
}

# Function takes in doc and returns a vector of dates:
# (Added NA check just in case)
Date = function(doc){
  dates = getNodeSet(doc, "//div[@class = 'user-action-time']/span")
  dates = sapply(dates, function(d){
    res = xmlGetAttr(d, "title")
    if (length(res) == 0){
      NA
    }
    else{
      res
    }
    })
  return(dates)
}

# Function takes in doc and returns a vector of reputation scores:
# (Added NA check just in case)
Rep = function(doc){
  all = getNodeSet(doc, "//div[@class = 'user-details']")
  reps = sapply(all, function(a){
    r = xmlChildren(a)$div
    r = xmlChildren(r)$span
    r = xmlValue(r)
  })
  return(reps)
}

# Function takes in doc and returns a vector of the number of views:
Views = function(doc){
  views = getNodeSet(doc, "//div[@class = 'views ']")
  views = sapply(views, xmlValue, trim = TRUE)
  views = lapply(views, function(v) strsplit(v[1], " ")[[1]][1])
  views = unlist(views)
```

```
  return(views)
}

# Function takes in doc and returns a vector of the number of answers:
Answers = function(doc){
  answers = getNodeSet(doc, "//div[@class = 'status answered'  or @class = 'status unanswered'
or @class = 'status answered-accepted' ]")
  answers = sapply(answers, xmlValue, trim = TRUE)
  answers = as.numeric(lapply(answers, function(a) substring(a, 1, 1)))
  answers = unlist(answers)
  return(answers)
}

# Function takes in doc and returns a vector of the vote scores:
Vote = function(doc){
  votes = getNodeSet(doc, "//div[@class = 'votes']")
  votes = sapply(votes, xmlValue, trim = TRUE)
  votes = lapply(votes, function(v) strsplit(v[1], '\r')[[1]][1])
  votes = unlist(votes)
  return(as.numeric(votes))
}

# Function takes in doc and returns a vector of URLs:
URL = function(doc){
  urls = getNodeSet(doc, "//div[@class = 'summary']/h3/a")
  urls = sapply(urls, xmlGetAttr, "href")
  urls = sapply(urls, function(i) sprintf("http://stackoverflow.com%s", i))
  return(urls)
}

# Function takes in doc and returns a vector of IDs:
ID = function(doc){
  ids = getNodeSet(doc, "//div[@class = 'question-summary']")
  ids = sapply(ids, xmlGetAttr, "id")
  ids = lapply(ids, function(i) substring(i, 18, 25))
  ids = unlist(ids)
  return(as.numeric(ids))
}

# Function that takes in doc and returns a vector of tags separated by "; ":
```

```
Tags = function(doc){
  tags = getNodeSet(doc, "//div[contains(@class, 'tags ')]")
  tags = sapply(tags, xmlGetAttr, "class")
  tags = lapply(tags, function(t){
    tg = strsplit(t, " ")
    tg = tg[[1]][-1]
    tg = gsub("t-", "", tg)
    tg = paste(tg, collapse = "; ")
  })
  tags = unlist(tags)
  return(tags)
}

# Function that takes in doc and returns a vector of posters:
# (Added in NA check to be safe)
Poster = function(doc){
  posters = getNodeSet(doc, "//div[@class = 'user-details']")
  posters = sapply(posters, function(p){
    res = xmlValue(p, trim = TRUE)
    res = strsplit(res, "\r")
    res = res[[1]][1]
    if(length(res) == 0){
      NA
    }
    else{
      res
    }
    })
  return(posters)
}

OnePage = function(doc){

  titles = Title(doc)
  answers = Answers(doc)
  dates = Date(doc)
  reps = Rep(doc)
  views = Views(doc)
  votes = Vote(doc)
  ids = ID(doc)
```

```
  urls = URL(doc)
  posters = Poster(doc)
  tags = Tags(doc)
  scrape = data.frame(title = titles, numAnswers = answers, date = dates, reputationScore
                  = reps, numViews = views, numVotes = votes, id = ids, url = urls,
                  poster = posters, tags = tags, row.names = NULL)

 return(scrape)
}

# Part 3 Analyzing R Questions on StackOverflow
# ===================================
load("~/UC Davis/STA 141/HW 6/rQAs.rda")

# Question 1
# --------------
# What is the distribution of the number of questions each person answered?
aOrC = subset(rQAs,( (rQAs$type == "answer") | (rQAs$type == "comment")))
A = subset(rQAs, rQAs$type == "answer")
par(mfrow = c(2, 1))
hist(table(aOrC$user), main = "Frequency of The Number of Comments or Answers By User",
    xlab = "Number of Comments or Answers", xlim = c(0, 1000), ylim = c(0, 6000))
hist(table(A$user), main = "Frequency of The Number of Answers By User",
    xlab = "Number of Comments", xlim = c(0, 1000), ylim = c(0, 6000))

# Question 2
# --------------
# What are the most common tags?
tags = res$tags
tags = as.character(tags)
tags = lapply(tags, strsplit, "; ")
tags = unlist(tags)
head(sort(table(tags), decreasing = TRUE), 10)

# Question 3
# --------------
# How many questions are about ggplot?
# Just looked at table from Q2

# Question 4
```

```
# ----------
# How many questions involve XML, HTML or Web Scraping?
rx = "xml|html|web scraping"
rQAs$q4 = grepl(rx, rQAs$text, ignore.case = TRUE)
# Need unique question ids:
tableq4 = table(rQAs$qid, rQAs$q4)
sum(tableq4[, 2] > 0) # 1327

# Question 5
# ----------
# What are the names of the R functions referenced in the titles of the posts?
rx = "[a-z][a-zA-z]{3,19}"
titles = res$title
titles = as.character(titles)
matches = gregexpr(rx, titles)
whatMatched = regmatches(titles, matches)
words = unlist(whatMatched)
words = unique(words)

# Question 6
# ----------
# What are the names of the R functions referenced in the accepted answers and comments of
# the posts? We can do better than we did in processing the title of the posts as there is
# HTML markup and we can find content in code blocks.
aOrC = subset(rQAs,( (rQAs$type == "answer") | (rQAs$type == "comment")))
rx = "[a-z][a-zA-z]{3,19}[(]"
match = gregexpr(rx, aOrC$text)
whatMatch = regmatches(aOrC$text, match)
functs = unlist(whatMatch)
functs = unique(functs)
```