Shirley Chew

STA 141

December 9, 2015

Homework #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.

X_____

Part I:

Strategy:

A. Create small functions for each of the things that need to be extracted.

1. Who posted it

- accommodates for "community" posts
- accommodates for deleted and anonymous users

i. Extract children from any node containing "user-info" because every post contains this (so that there are 50 posts if the number post per page is set to 50)

ii. Find the valid children nodes (not "community", anonymous, or deleted posts)

a. Loop through each child node found from "user-info" path

- if children under "user-details" is more than one, it is a "community" post
- if "user-details" only has one child,
  - if the children under "user-details" is not at least 2, then that post is either anonymous or deleted username, set to TRUE
  - if children under "user-details" is less than two, set to FALSE

iii. Create a data frame with "NAs"

iv. Make a path for valid posts, to get usernames

v. Extract the usernames and replace the "NAs" with the valid post usernames

2. When it was posted

- accommodates for "community" posts that do not have a time posted
- anonymous and deleted posts have a time posted

i. Extract children from any node containing "user-info" because every post contains this

ii. Find the valid children nodes (not "community" posts)

a. Loop through each child under "user-info" to

- Extract the number of "user-details" children
  - if more than one child, then it is a "community" post, set to FALSE
  - if only one child, it is a regular post, then set to TRUE

iii. Make a path for valid posts, to get time posted

iv. Create a data frame with "NAs"

v. Extract the usernames and replace the "NAs" with the valid post usernames

3. **The title of the post**
    i. Find path to the title
    ii. Extract the text
    iii. Store in a dataframe
4. **The reputation level of the poster**
    - accommodates for "community" posts (don't have reputation score)
    - accommodates for deleted and anonymous users (don't have reputation scores)
    - accommodates for when reputation score changes in to "#k" (ex/ 5k) reputation, take reputation score from title attribute, otherwise, take from the text

    i. Extract children from any node containing "user-info" because every post contains this
    ii. Find the valid children nodes (not "community", anonymous, or deleted posts)
        a. Loop through each child node found from "user-info" path
            - if children under "user-details" is more than one, it is a "community" post
            - if "user-details" only has one child,
                - if the children under "user-details" is not at least 2, then that post is either anonymous or deleted username, set to TRUE
                - if children under "user-details" is less than two, set to FALSE

    iii. Loop through valid posts to get the reputation score
        a. Get path to the reputation score text
        b. If the extracted reputation score contains letter "k", then make path to title attribute and get reputation score from there instead
            - Replace every instance of letters, comma, and white space to get only the score from "reputation score xxxxx"
        c. if extracted reputation score doesn't contain a "k", then remove all commas from the text extracted score
    iv. Create a data frame with "NAs"
    v. Get reputation scores and replace the "NAs" with the valid post usernames
5. **The current number of views for the post**
    i. accomodates for class change when number of views becomes 1k+ (ex/ "views hot", "views warm", "views supernova", etc.)
    i. Get path to title attribute for number of views
    ii. Extract the text from title attribute
    iii. Substitute all instances of letters, commas, and spaces from "xx,xxx views"
    iv. Store in a data frame
6. **The current number of answers for the post**
    i. accommodates for class difference of "status answered-accepted", "status unanswered", "status answered", etc.
    i. Get path to the text of current number of answers
    ii. Extract the text
    iii. Store the info into a data frame
7. **The vote score for the post**

i. accounts for the class change when votes become high, such as "votes-count-post high-scored-post", "vote-count-post", etc.
i. Get path to the text of current number of answers
ii. Extract the text
iii. Store the info into a data frame

8. **The URL for the page with the post, answers, and comments**
    i. Get the path to the url at attribute href
    ii. Extract the text from the attribute
    iii. Use extracted text to convert to the full url
    iv. Store the info into a data frame
9. **The id uniquely identifying the post**
    i. Get the path of the unique id from at attribute "id"
    ii. Extract the text from the attribute
    iii. Remove every instance of letters and dashes from "question-summary-xxxxx" to get the unique id
10. **The tags for each post**
    i. Get path for div tags starting with class = 'tags' to get EACH post's tags
    ii. Loop through EACH post to:
        a. Find all "<a> tags" with class = "post-tags" to get all the tags in each post
        b. Extract the text for each tag
        c. Paste all the tags together, separated by a space (" ")
    iii. Store all tags into a data frame of 1 column

B. Make an encompassing function to bind all small functions together for ONE page

1. The parameters are html and url
2. Calls the smaller functions:
    i. username()
    ii. time()
    iii. title()
    iv. reputation()
    v. views()
    vi. answers()
    vii. votes()
    viii. post_url()
    ix. post_id()
    x. tags()

    and stores each of them into a variable
3. Column bind each of the functions to make a data frame of 10 variables

C. Make a top-level function that loops over one page and gets the url for the next page until limit reached

1. Parameter "pageLimit" default value set to an arbitrarily large value (infinty)

2. Create an empty list to hold the data frames for EACH page
3. Keep a pageCount, initially set to 1
4. Loop over each page until page limit reached to:
    i. get the html of the page
    ii. store the data frame for the current page into the list
    iii. if the page limit is reached, break the loop
    iv. if page limit is not reached,
        a. update the url to the next page
            - make a function that finds the next page
        b. update the page count by one iteration
5. Combine the list of data frames for EACH PAGE into one data frame

I showed that these functions work by extracting the information from the first 2000 pages:

```
> nrow(first_2000_pages)
[1] 100000
> ncol(first_2000_pages)
[1] 10
```

first_2000_pages 100000 obs. of 10 variables

Snippet of what the first 5 posts look like:

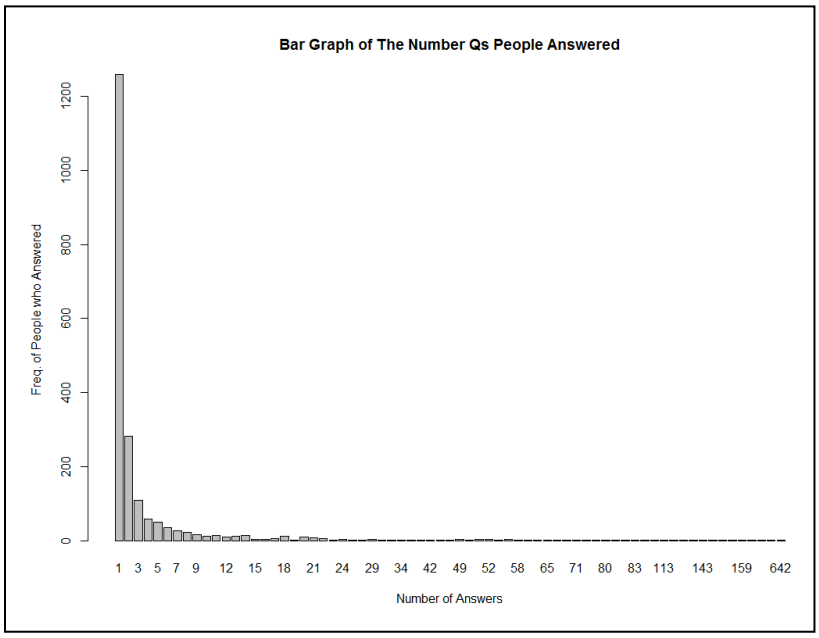
	times	post_title	views	answers	votes	post_urls	post_id	tags	user_name	rep
1	2015-12-07 06:51:26Z	R: How to get a value in data frame column dependin...	6	0	0	http://stackoverflow.com/questions/34127959/r-h...	34127959	r	Michael Simonsen	13
2	2015-12-07 06:45:41Z	bnlearn wrong dimensions for node	5	0	0	http://stackoverflow.com/questions/34127885/bnl...	34127885	r machine-learning bayesian-networks	Woody1193	113
3	2015-12-07 06:21:19Z	Reactive Table Not Showing up in Shiny	12	0	0	http://stackoverflow.com/questions/34127605/rea...	34127605	r shiny shinydashboard	Akshit	1
4	2015-12-07 06:15:32Z	R: Show grids and rotate axis text on a 3D barplot usi...	6	0	0	http://stackoverflow.com/questions/34127531/r-s...	34127531	r 3d grid axes	DaniCee	232
5	2015-12-07 06:12:20Z	Fill in the values based on first cell value	13	1	0	http://stackoverflow.com/questions/34127497/fill-i...	34127497	r	Jay khan	83

The Pros of my strategy is that it can account for many different posts because anything that is not similar to regular posts are considered invalid posts and automatically set to NA. The Cons of this strategy is that if the "user-info" was not included in one post, some of my small functions would not work, and thus will not create a data frame (error). The code is dependent on all of the small functions extracting the same amount of posts per page.

Part III

Question 1:

The distribution is skewed to the right. On the x-axis is the number of answers and on the y-axis is the frequency of people who answered that many times. Most people answered only once for all questions, and many posters answered only a few times (1, 2, or 3 times).



Question 2:

Using the function I created in part I, I extracted 2000 pages of 50 post per pages into a data frame to get 100,000 observations (or posts). This is the data frame I used for this question to get the tags. They are all tags related to r because the default tag is r and they are all quite common and difficult topics, so it is no surprise that they are one of the most tagged.

The ten most common tags were:

regex, rstudio, dplyr, matrix, data.table, shiny, plot, data.frame, ggplot2, and r

regex	rstudio	dplyr	matrix
1721	1889	1977	2641
data.table	shiny	plot	data.frame
3067	3127	4811	4883
ggplot2	r		
8543	100000		

Question 3:

There are 959 questions that are about ggplot.

I used the data frame given by Duncan for part 3 to subset the types of posts that were questions and then found every instance of "ggplot" in the text no matter if they were lowercase or uppercase.

FALSE	TRUE
9045	959

Question 4:

There are 721 questions about "XML"
There are 1149 questions about "HTML"
There are 7 questions about "Web Scraping"

```
[[1]]

FALSE  TRUE
 9283   721

[[2]]

FALSE  TRUE
 8855  1149

[[3]]

FALSE  TRUE
 9997     7
```

I used the data frame given by Duncan for part 3 to subset the types of posts that were questions and then found every instance of "XML", "HTML", and "Web Scraping" in the text no matter if they were lowercase or uppercase.

Question 5:

Strategy:

1. Get URL from the row names of the data frame
2. Get the title from the URL
3. Loop to split the list of words in title by dashes so the title is in words
4. Unlist the title words
5. Get the most common word in the list to see if the top 2000 are functions
6. Get the base function names to compare to common title names

Among the most common functions in the titles are while(), table(), plot(), and legend().

A snippet of the most common functions:

```
"levels"        "grouped"       "across"          "empty"
"c"             "creating"      "grid"            "intervals"
"chart"         "user"          "facet"           "nested"
"according"     "objects"       "r.title2, how"   "script"
"merge"         "expression"    "point"           "some"
"install"       "strings"       "id"              "sample"
"given"         "histogram"     "doesnt"          "like"
"xml"           "many"          "long"            "matching"
```

Question 6:

We can do better than processing the title of the posts because there is HTML markup, we can find the accepted answers comment and other comments and usually, functions have parenthesis, so it is much more easily found than when searching in the title where people may not put parenthesis on their functions. Also, we can check for comments in the HTML that have user-made functions because those always start with the word "function(...)".