STA141 Assignment VI

Weitong(Jessie) Lin

December 8, 2015

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.

```
# load packages, set directory
library(XML)
library(RCurl)
setwd('/Users/RayLJazz/Dropbox/UC Davis/141/web scraping')
```

part 1

My strategy for this assignment is to first extract every information that we want for each post, then go through the whole page, finally go to the number of page we want.

First, let see 10 functions which are used to extract the specific info that we want. The input for each functions are xml node chunk for each post. The output is the info we want.

- **1.id** My strategy for id is to get the 'id' attribute value of the top parent for eacn post chunk. Then remove 'question-summary' and get the id number

```
each_ids = function(each_post){

  # get 'id' attribute content
  ids = xmlAttrs(each_post)['id']
  # remove letters, id number left
  gsub('[-a-zA-z]*', '', ids)

}
```

- **2.date**

My strategy here is to extract div tag 'user-action-time', followed by span tag with class 'relativetime'. Then the date is in the 'title' attribute.

```
each_dates = function(each_post){
  # xpath
  xpath_date = ".//div[@class = 'summary']
              //div[@class = 'user-action-time']
              /span[@class = 'relativetime']"
  # get the node
  date_node = getNodeSet(each_post, xpath_date)
  # find the content of title attribute
  xmlAttrs(date_node[[1]])['title']

}
```

- **3.tags**

My strategy here is to find 'a' tag with class 'post-tag'. All tags are showed here. Then get every possible tags and combine them into a string.

```
each_tags = function(each_post){

  xpath_tags = ".//div[@class = 'summary']
                //a[@class = 'post-tag']"
  # get all exist tags
  tag = xpathSApply(each_post, xpath_tags,
              xmlValue, trim = TRUE)
  # combine into string by ';'
  paste(tag, collapse = '; ')
}
```

- **4.title**

My strategy here is to extract h3 tag followed by 'a' tag with class 'question-hyperlink', the content is the titles

```
each_title = function(each_post){
  xpath_title = ".//div[@class = 'summary']
                 //h3/a[@class = 'question-hyperlink']"
  # get title
  xpathSApply(each_post, xpath_title,
              xmlValue, trim = TRUE)
}
```

- **5.url**

My strategy here is to extract 'a' tag with class 'question-hyperlink', then get the link in 'href'. after getting the link, combine the link with the main link

```
each_urls = function(each_post){
  xpath_urls = ".//div[@class = 'summary']
                //h3/a[@class = 'question-hyperlink']/@href"
  # get the link
  urls = getNodeSet(each_post, xpath_urls)[[1]][[1]]
  # set base link
  baseURL = 'http://stackoverflow.com'
  # combine the question link and the base link to get the complete link
  getRelativeURL(urls, baseURL)
}
```

- **6.views**

My strategy here is two part. First is the common routine that extract div tag with class 'statscontainer' and then div tag with class 'views'. Then we get the views number. However, When I scrape some pages, I find that some hotted posts, which have been searched and seen for a lot of time, will be assigned as a 'warm view post'. Thus, the **class** will be changed into 'views warm', that the color will be different. So here I write that

if we couldn't exract anything under class 'views' pattern, then go to try class 'views warm' pattern. My cons here is that I only scraped 10000 posts. If there are more 'view' class type shows up, I will need to find them one by one and then update the new patterns. Maybe regular expression will avoid this since we can get the class which contains the word 'views'. However it will also have some risks that we extract other things.

```
each_views = function(each_post){
  xpath_views = ".//div[@class = 'statscontainer']
                 //div[@class = 'views ']"
  # get views number
  views = xpathSApply(each_post, xpath_views,
              xmlValue, trim = TRUE)
  # if we get nothing from the 1st pattern, try 2nd pattern
  if (length(views) == 0) {
    # second patter
    xpath_views = ".//div[@class = 'statscontainer']
                 //div[@class = 'views warm']"
  views = xpathSApply(each_post, xpath_views,
              xmlValue, trim = TRUE)
  }
  # get ride of 'views'
  gsub('[ ::alpha::].*', '', views)
}
```

- **7.votes**

My strategy here is to extract div tag with class 'statscontainer' and then div tag with class 'vote-count-post'. The same idea with the 'views', the page I scrape is the latest 10000 posts. So there are basicly less votes yet. If we go through some old classic posts, there may be some posts with high votes will use another class, I guess is 'vote-count-post warm' or something like that. I can't prove whether this situation exists or not. So this is a kind of cons for my strategy. Maybe regular expression will avoid it.

```
each_votes = function(each_post){
   xpath_votes = ".//div[@class = 'statscontainer']
                  //span[@class = 'vote-count-post ']"
   # get votes number
   xpathSApply(each_post, xpath_votes,
              xmlValue, trim = TRUE)
}
```

- **8.answers**

My strategy here is to extract div tag with class 'statscontainer' and then div tag with 3 type of class 'status answered', 'status unanswered' and 'status answered-accepted'. The same idea with the 'views', the page I scrape is the latest 10000 posts. So there are basicly less answers yet. If we go through some old classic posts, there may be some posts with high answers will use another class, I can't prove whether this situation exists or not. So this is a kind of cons for my strategy. Maybe regular expression will avoid it.

```
each_answers = function(each_post){
  xpath_answers = ".//div[@class = 'statscontainer']
                   //div[@class = 'status answered'
                      or @class = 'status unanswered'
                      or @class = 'status answered-accepted']/strong"
```

```
  # get answers number
  xpathSApply(each_post, xpath_answers,
              xmlValue, trim = TRUE)
}
```

- **9.user**

My strategy here is also two part. First extract div tag with class 'user-details' and then followed by 'a' tag. However, not all user have a hyperlink for it's user profile, so tag 'a' may not exist. So I will first try "with 'a' tag" strategy, if get nothing, then try "without 'a' tag" strategy

```
each_user = function(each_post){

  xpath_user = ".//div[@class = 'summary']
                //div[@class = 'user-details']/a"
  # try the 1st pattern
  user = xpathSApply(each_post, xpath_user,
              xmlValue, trim = TRUE)

  # if we get nothing, try 2nd pattern
  if (length(user) == 0) {
    xpath_user = ".//div[@class = 'summary']
                //div[@class = 'user-details']"
    user = xpathSApply(each_post, xpath_user,
              xmlValue, trim = TRUE)
  }
  # return user
  return (user)
}
```

- **10. reputation score**

My strategy here is to extract div tag with class 'user-details' and then span tag with class 'reputation-score'. However, later on I find that the user who don't sign up to this website will not have the 'reputation-score'. So the class will not exist. I decide to asign 'NA' to this user as reputation-score.

```
each_reputation = function(each_post){

  xpath_reputation = ".//div[@class = 'summary']
                      //div[@class = 'user-details']
                     //span[@class = 'reputation-score']"
  # get score
  score = xpathSApply(each_post, xpath_reputation,
              xmlValue, trim = TRUE)
  # if we got nothing, then 'NA'
  if(length(score) == 0) {
    score = NA
  }
  return (score)
}
```

- **scrape one post**

This function is to scrape all above info and then combine them into a data.frame. The input is the xml node chunk for each post. The out put is a data frame with all info for that post

```
scraping_onepost = function(each_post){

  # scrape all info
  id = each_ids(each_post)
  date = each_dates(each_post)
  tags = each_tags(each_post)
  title = each_title(each_post)
  url = each_urls(each_post)
  views = each_views(each_post)
  votes = each_votes(each_post)
  answers = each_answers(each_post)
  user = each_user(each_post)
  reputation = each_reputation(each_post)
  # combine into data.frame
  data_onepost = data.frame(id, date, tags, title, url, views,
                            votes, answers, user, reputation,
                            stringsAsFactors = FALSE)
  # get rid of rownames
  row.names(data_onepost)=NULL
  # return data frame
  return (data_onepost)
}
```

- **next URL**

This function is to get the next page's hyperlink for each page. The input is the current hyperlink. The output is the updated link for next page.

```
getNextURL =
function(html_url)
{
    # parse the link
    html_stackR = htmlParse(html_url, trim = TRUE)
    # find the next page's link
    nxt = unique(unlist(getNodeSet(html_stackR, "//a[@rel = 'next']/@href")))
    # combine with the base link, get the completed link
    getRelativeURL(nxt, docName(html_stackR))
}
```

- **scrape one page**

This function is to scrape the whole page's info. The input is the page link that we want to scrape. The output is the dataframe of all info for this page

```
scraping_onepage = function(html_url){
    #parse the link
    html_onepage = htmlParse(html_url, trim = TRUE)
    # scrape the whole page, one by one post
    one_page = xpathApply(html_onepage,
```

```
                        "//div[@class = 'question-summary']",
                        scraping_onepost)
    # rind all posts info
    curr_page = do.call(rbind, one_page)
    return (curr_page)
}
```

- **scrape the whole website**

This function is to scrape as many pages as the caller want. The input is the website's main link, the tag you want to see, like'r' or 'python', and then how many page you want to scrape. Also, if caller is not assigning the number of page, then it will scrape all pages. Also, if we can not get the next page, it means that it's the last page, then the function will stop. Also, I define each page has 50 posts

```
scraping_allpage = function(html_url, tag, num_page){

  # if not assing num_page, set it as Inf number, so it will scrape all
  if (missing(num_page)){
    num_page = Inf
  }
  # get the first link
  html_url = paste0(html_url, "tagged/", tag, "?sort=newest&pagesize=50")
  all_page = NULL
  page_num = 1
  # when page_num is smaller than we want, loop
  while(page_num <= num_page){
    # scrape current page
    curr_page = scraping_onepage(html_url)
    # combine with the existing data.frame
    all_page = rbind(all_page, curr_page)
    # get the next page link
    html_url = getNextURL(html_url)
    # if we didn't get the next link, jump out while loop
    if (length(html_url) == 0)
      break
    page_num = page_num + 1
  }
  return (all_page)

}
```

Now let's see whether my 'scraping_allpage' function will work or not. I will ask the function to scrape the **stackoverflow** wensite with tag **r**. And I want to **200** pages. Since there are 50 posts each page, so totally I will get 10000 posts info.

```
html_url = "http://stackoverflow.com/questions/"
tag = 'r'
page = 200
page_200 = scraping_allpage(html_url, tag, page)
dim(page_200)
```

```
## [1] 10000    10
```

From the dimension, we can see the I get 10000 posts with 10 info, which is right with what I want.

Now let's see the first 6 post's info, which looks right.

```
head(page_200)
```

```
##         id                 date              tags
## 1 34189367 2015-12-09 21:30:44Z      r; lubridate
## 2 34189364 2015-12-09 21:30:18Z                 r
## 3 34189288 2015-12-09 21:24:48Z         r; matlab
## 4 34189264 2015-12-09 21:23:11Z r; shiny; leaflet
## 5 34189228 2015-12-09 21:21:12Z                 r
## 6 34189185 2015-12-09 21:19:11Z          r; shiny
##                                                 title
## 1    Find day of year with the lubridate package in R
## 2 Best ways to visualize multivariate data with gaps?
## 3                          transfer MATLAB data into R
## 4            Reactive Data Frame in Shiny Leaflet Map
## 5           R get string of unique vales from Titanic
## 6                Shiny: passing values to c function
##                                                                                                 url
## 1   http://stackoverflow.com/questions/34189367/find-day-of-year-with-the-lubridate-package-in-r
## 2 http://stackoverflow.com/questions/34189364/best-ways-to-visualize-multivariate-data-with-gaps
## 3                        http://stackoverflow.com/questions/34189288/transfer-matlab-data-into-r
## 4           http://stackoverflow.com/questions/34189264/reactive-data-frame-in-shiny-leaflet-map
## 5          http://stackoverflow.com/questions/34189228/r-get-string-of-unique-vales-from-titanic
## 6                http://stackoverflow.com/questions/34189185/shiny-passing-values-to-c-function
##   views votes answers             user reputation
## 1     4     0       0 Joshua Rosenberg        164
## 2     5     0       0             Juan          1
## 3     7     0       0            JC Ma          1
## 4     4     0       0         medavis6         97
## 5     8     0       0     Mario Trento         88
## 6     4     0       0             Matt         18
```

Just only for this 6 posts, we can still find that the user 'Ronak Shah', who has the highest reputation '1886', also got the highest question views, and also got the answer back, also got a positive vote which can indicate that this is a good question or we already got a good answer. Thus, we can roughly conclude that, more the reputation score you have, you may have more views and more chance to get the answer to solve your problem. Thus, try to contribute more on stackoverflow. We need to involve in, not just seek answer and never answer questions even if you know the answer. If you contribute more, then you will receive more. (I guess it's the same idea with Piazza).

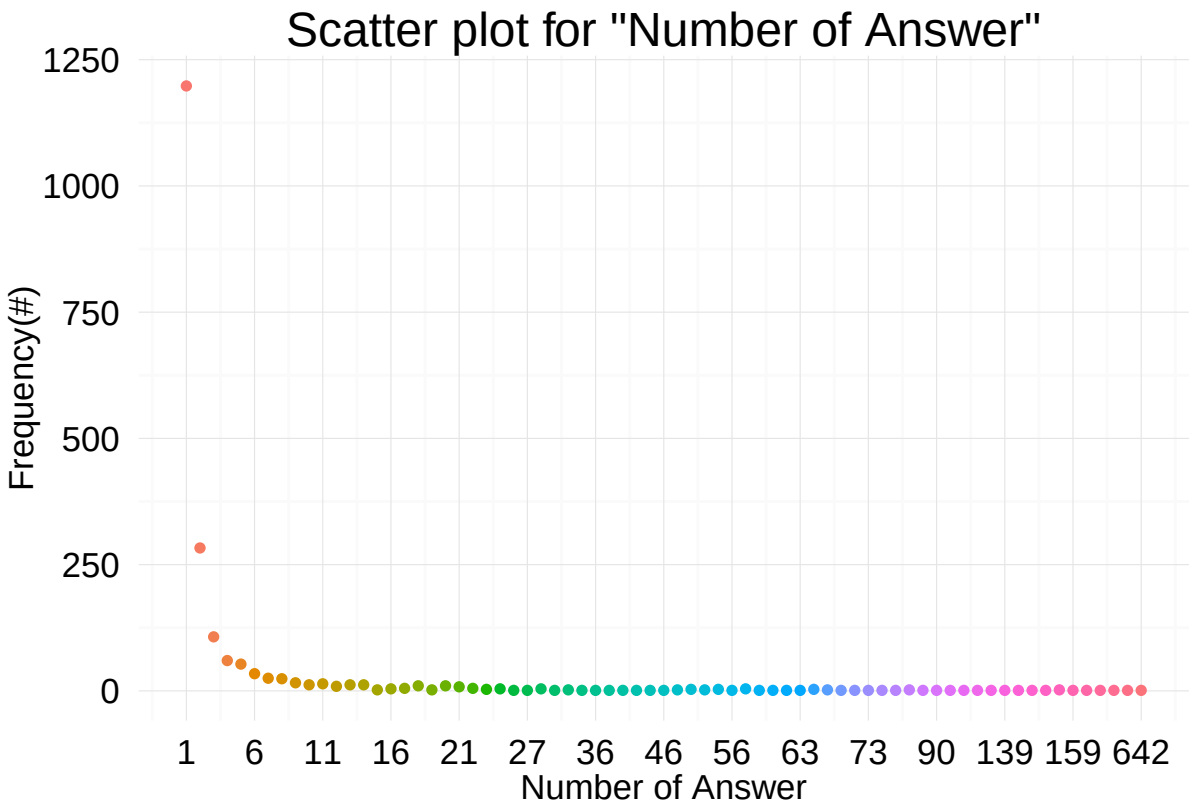
Also, for these only latest 6 r-taged posts, we can still see that there are also some tags like 'ggplot' for **data visualization**, 'dplyr' for **data pre-processing** and 'json' for **web scraping**. This can prove that how hot these three topics are. This situation will also be proved in the later on analysis.

part 3.

- **1.What is the distribution of the number of questions each person answered?**

Here I will use Duncan's dataset. My strategy is first subset the posts with only type = 'answer'. Then to see the frequency of num of answer.

```r
# load Duncan's dataset
load('rQAs.rda')
# only get the 'answer' type data
only_answer = subset(rQAs, rQAs$type == 'answer')
# count number of answer for each user
num_answer_user = as.data.frame(table(only_answer$user))
# to count the frequency for each number of answer
freq_num_answer = as.data.frame(table(num_answer_user$Freq))
colnames(freq_num_answer) = c('num_ans', 'Frequency')
# draw plot
library(ggplot2)
ggplot(freq_num_answer, aes(x = as.numeric(num_ans), y = Frequency)) +
  # data mapping
  geom_point(aes(color = factor(num_ans)), alpha = 1, size = 2) + # draw a scatter plot
  # plot labeling
  scale_x_continuous('Number of Answer',
                     breaks = c(1,6, 11,16, 21,26, 31,36, 41,46, 51, 56, 61,66,71),
                     labels = levels(factor(freq_num_answer$num_ans))
                     [c(1,6, 11,16, 21,26, 31,36, 41,46, 51, 56, 61,66,71)]) +
  scale_y_continuous('Frequency(#)') +
 labs(title = 'Scatter plot for "Number of Answer"')+
  # theme elements
  theme_minimal() +
  theme(
    legend.position = "none",
    axis.text.x = element_text(size = 13),
    axis.text.y = element_text(size = 13),
    title = element_text(size = 15),
    axis.title = element_text(size = 13),
    axis.ticks = element_blank())
```



From the plot, we can see that most user are only answer once or twice, which indicate that most of the user may more like to seek the answer, but not answer the questions.

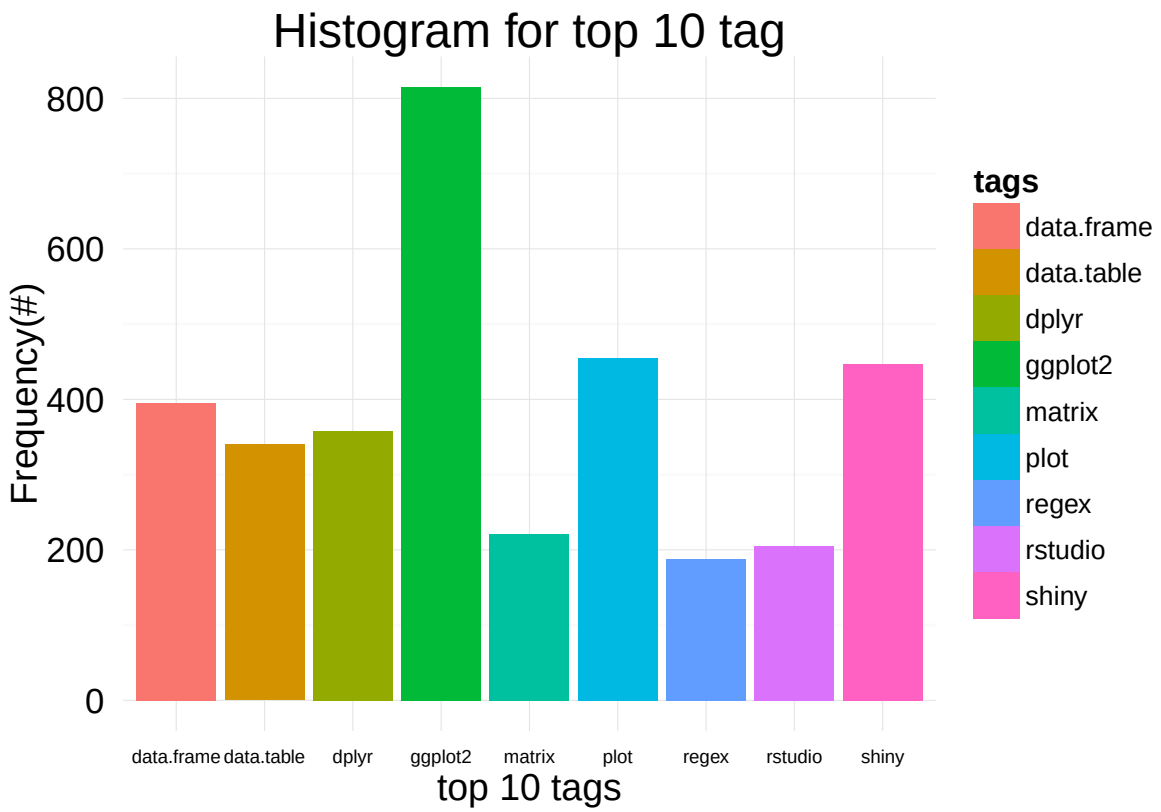
- **2. What are the most common tags?**

I will use my 10000 posts dataset in this question. My strategy here is to first split the tags for each post by ';', then combine them all to count the unique tag.

```
# get the tag column from the part one
tags = page_200$tags
# split each data by ';', then we have all tags
all_tags = unlist(strsplit(tags, ';'))
# remove whitespace
all_tags = gsub('[ ]*', '', all_tags)
# change tag as factor
all_tags = factor(all_tags)
# count each tag
all_tags = as.data.frame(table(all_tags))
# rank the tag
tags_rank = all_tags[order(all_tags$Freq, decreasing = TRUE),]
# show the top 10 tags
top_1o_tags = head(tags_rank, 10)
top_1o_tags
```

```
##        all_tags  Freq
## 1314          r 10000
## 628     ggplot2   815
```

```
## 1227       plot   455
## 1553      shiny   447
## 353  data.frame   395
## 440       dplyr   358
## 354  data.table   340
## 977       matrix   221
## 1483    rstudio   205
## 1390      regex   188
```



From the result, we can see that, not suprisingly, we get 10000 tag 'r', since what I scrap in the part one is 10000 posts which has tag 'r'. This can be another way to prove that the thing I scrape from part one is right. Also, from the plot, we can see that 'ggplot' tag holds the top 2 position. Here we can find that 'plot' has half less tags than 'ggplot', which means that 'ggplot', a powerful plotting package, becomes more and more popular than 'plot' since these 10000 posts are the recent posts on StackOverflow. Also, 'shiny' also becomes a very hot topic because its function: 'A web application framework for R'. What's more, 'dplyr' is also very hot since it's quite powerful for data pre-processing. From this question, we can find what the hot topics are for R recently.

- **3. How many questions are about ggplot?**

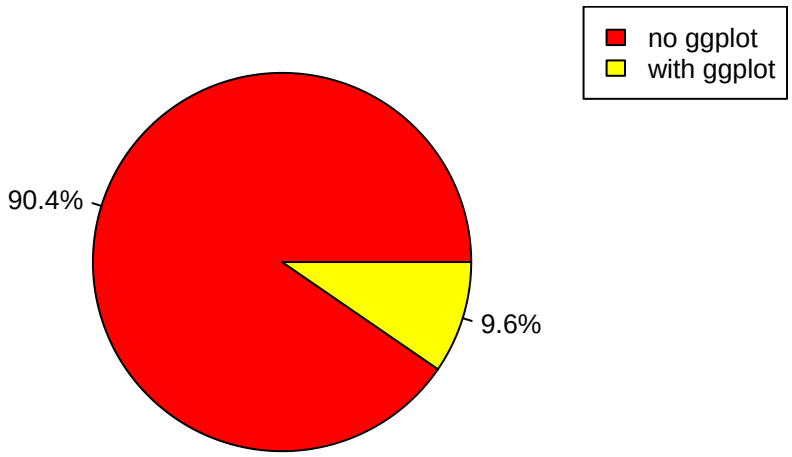
In this question I will be back to use Duncan's dataset. To see how many question are about ggplot, my strategy is to scrape 'ggplot' in text, but only in 'question'. It may also a good choice to scrape from both question and answer and commend, but it may be the situation that the question is not about 'ggplot', but later on the it starts to turn to 'ggplot' topic. So in this question I will scrape from 'question' type of data.

```
# get only question data
Question_data = rQAs[rQAs$type == 'question',]
```

```r
# get all text
texts = Question_data$text
# keyword: ggplot
rx_ggplot = "ggplot"
# to see which observation matches this first strategy
ggplot_count1 = grepl(rx_ggplot, texts,ignore.case = TRUE)
# To see how many observations match this first strategy
table(ggplot_count1)
```

```
## ggplot_count1
## FALSE  TRUE
##  9045   959
```

Piechart for keyword:'ggplot' in post



Here we can see that out of 10004 questions, there are 959 questions are about ggplot. Around 9.6% questions are about 'ggplot'. This is a quite large percent since R can do such a lot of things. This indecate that ggplot is really hot topic and more and more people would like to use it. Thus, a lot of questions about 'ggplot' will come out.

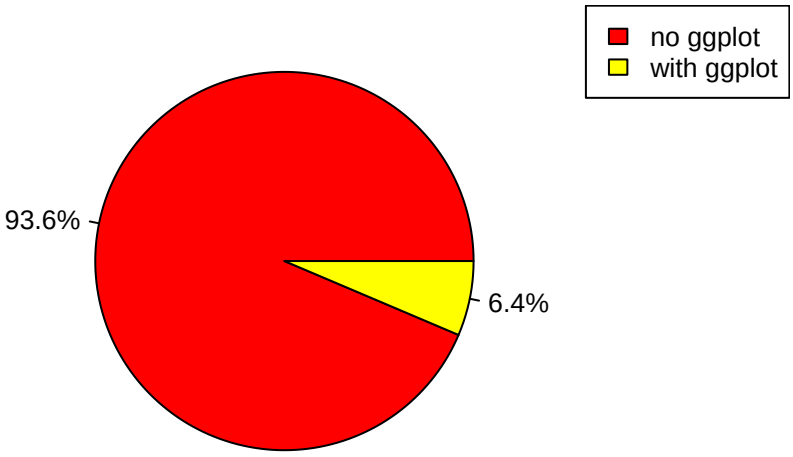
- **4. How many questions involve XML, HTML or Web Scraping?**

In this question I will be back to use Duncan's dataset. The same idea with Q3, o see how many question are about XML, my strategy is to scrape 'XML' topic in text, but only in 'question'. It may also a good choice to scrape from both question and answer and commend, but it may be the situation that the question is not about 'XML', but later on the it starts to turn to 'XML' topic. So in this question I will scrape from 'question' type of data.

```r
# for XML, HTML or Web scraping
rx_Web_scraping = "Web scraping|xml|html"
# to see which observation matches these strategy
web_count = grepl(rx_Web_scraping, texts,ignore.case = TRUE)
# To get the id which observations match these strategy
web_id = Question_data[web_count == TRUE,'id']
# get the unique question id
unique_id = unique(web_id)
# show how many
length(unique_id)
```

```
## [1] 639
```

Piechart for keyword:web scraping in post



Here we can see that out of 10004 questions, there are 639 questions are about web scraping related terms. Around 6.4% questions are about web scraping related terms. This is a quite large percent since R can do such a lot of things. This indicate that web scraping is really hot topic and more and more people would like to use it. Thus, a lot of questions about it will come out. Web scraping is really the hot topic nowadays! Thanks Duncan for provideing as such a powerful R packages.

- **5. What are the names of the R functions referenced in the titles of the posts?**

In this question I will use several packages which are hot recently from the above result. 'shiny', 'ggplot2', 'dplyr', 'XML' and 'base' will be matched in this question. My strategy is to split the title name into each single word, and then to check if each single word is a function name by using 'is.function()'. Also, the title is extracted from the hyperlinks, which are the rownames for each question

```
# packages that I'm interesting
library(shiny)
library(ggplot2)
library(dplyr)
library(XML)
# get all link
links = rownames(Question_data)
# scrape title from link
pattern_link = 'http://stackoverflow.com/questions/[0-9]*/([-0-9A-Za-z]*)(.1)*'
title = gsub(pattern_link,'\\1',links)
# get the unique title
titles = unique(title)
# split the title to get each word
titles_word = strsplit(titles, '-')

# this function is to if each word are matched with function name
is_function = function(word){

    # if:
    #   - no match: return "NA"
    #   - has value: return that word
```

```
    if (is.function((get0(word, ifnotfound = NA)))) {
      return (word)
    }
  }

# this function is to get each post's matched function words
function_list = function(word_list){

    # go through each word in the word_list
    fun_word = unlist(sapply(word_list[[1]],
                             function(word) is_function(word)))
    # change the result into a combined string by';'
    paste(fun_word, collapse = ";")
}

# go through all posts to get matched word string
word_list = sapply(titles_word, function_list)
# get all single word
word_list = unlist(strsplit(word_list, ';'))
# count the word
count_function = as.data.frame(table(word_list))
# rank with decreasing order
function_title_rank = count_function[order(count_function$Freq, decreasing = TRUE),]
# show the top 10 functions
head(function_title_rank, 10)
```

```
##     word_list Freq
## 148      plot   75
## 99         is   63
## 83     ggplot   57
## 167    remove   51
## 45       data   49
## 70       find   42
## 82        get   42
## 210    subset   38
## 126     merge   35
## 200     split   35
```

From the result, we can see that 'plot' function appears the most. Also, 'ggplot', 'subset', 'merge' and 'split' also appear a lot. This result makes sense because **data visuallization** is quite a hot topic nowadays. More and more people would like to use plot to show a data analysis visually. So it's not surprising that 'plot' and 'ggplot' function appear a lot. Also, 'subset', 'merge' and 'split' are the powerful functions to pre-processing data. The idea of 'big data' is popular. More and more people will tend to deal with some high dimentional uncleaning datasets. So it's also not surpring that we get 'subset', 'merge' and 'split' on the top 10 list.

Pros and cons: For the functions that I talk above, they are quite reasonable. However, for the functions like 'is', 'get', 'data', we do have this kind of function. However, they are also common language words, which may appear with no function meanings. This is a con of my strategy. I guess using regular expression can avoid this kind of things, but at the same time may spend too much time and also may lose the information, for example, we do have a is() function in the title. Also, for this strategy, 'base' package will always be included. What if I don't want to see 'base'? I will use another strategy in Q6 which can avoid including 'base'.

- **6. What are the names of the R functions referenced in the accepted answers and comments of the posts?**

In this question I will still use several packages which are hot recently from the above result. 'shiny', 'ggplot2', 'dplyr' and 'XML' will be matched in this question. package 'base' will be excluded.

```
# packages that I'm interesting
package_name = c("ggplot2","shiny", 'dplyr','XML')
# function name pool
function_pool = ls(getNamespace(package_name), all.names=TRUE)

# split the title to get each word
texts_word = strsplit(texts, ' ')
function_list = function(word){
  is_in = word %in% function_pool
  fun_word = word[is_in == TRUE]
  # change the result into a combined string by';'
  paste(fun_word, collapse = ";")
}

# go through all posts to get matched word string
word_list = sapply(texts_word, function_list)
# get all single word
word_list = unlist(strsplit(word_list, ';'))
# count the word
count_function = as.data.frame(table(word_list))
# rank with decreasing order
function_command_rank = count_function[order(count_function$Freq, decreasing = TRUE),]
# show the top 10 functions
head(function_command_rank, 10)
```

```
##    word_list Freq
## 33    ggplot  208
## 15  distance  141
## 12    breaks  105
## 40     layer   63
## 68      xlab   56
## 71      ylab   49
## 73      ylim   45
## 10       bin   40
## 16     facet   39
## 42    limits   37
```

From the result, we can see that 'ggplot' function appears the most. Acctually most of them are the function for plotting. This indicate that **Data visualization** is really hot nowadays.

From this assignment, I really learn a lot about the developing trend for R. Like I said previously, **data visualization**, **data pre-processing** and **web scraping** are indeed the hot topics nowadays. These three skills even becomes a requirment for everyone. And also I realize that I have already dabbled these three skills in STA 141 class. I do appreciate that I can learn such a wonderful thing from this class, although I may not get a good grade. Although this is my last 141 assignment, I will not stop learning R. This class indeed draw my interest and passion to R and data science. Thanks Duncan for teaching us so many wonderful R! Thanks Nick, Michael and Yuki for grading our annoying paper and answering our questions patiently! Hope I will have the chance to take 141B and 141C before I graduate!