Assignment 6

Huang Fang 913439658

December 9, 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.

Part 1

Q1, Q2. Process the current summary page of posts, starting with the first page of results and extract information

Aftering parsing the first page.
`page = htmlParse("http://stackoverflow.com/questions/tagged/r?sort=newest&pagesize=50")`
We need to find the 50 nodes who represent questions, and find its corresponding infomation seperately.
To find 50 nodes, we use the pattern `"//div[@class = ('question-summary')]"`

Note that we are going to find the corresponding information, so we will always use sapply to loop over these 50 nodes.
For id, after getting the attribute "id", we have to use regular expression to get the part we want.
```
id = sapply(posts, function(x) gsub(".*?([0-9]+).*","\\1", xmlGetAttr(x, "id")))
```
For date, the pattern is:
```
date = sapply(posts, function(x) xpathSApply(x, ".//span[@class = 'relativetime']", function(y)
xmlGetAttr(y,"title")))
```
For user:
```
user = sapply(posts, function(x) xpathSApply(x, ".//div[@class = 'user-details']//a", xmlValue))
```
For title:
```
title = sapply(posts, function(x) xpathSApply(x, ".//div[@class = 'summary']//
        a[@class = 'question-hyperlink']", xmlValue))
```
For votes:
```
votes = sapply(posts, function(x) xpathSApply(x, ".//div[@class = 'votes']//strong", xmlValue))
```
For views, we can find 4 types of class, "views ", "views warm", "views hot" and "views supernova":
```
views = sapply(posts, function(x) xpathSApply(x, ".//div[@class = 'views ' or @class = 'views
supernova' or @class = 'views warm' or @class = 'views hot']", function(x) strsplit(xmlGetAttr(x,
"title"), " ")[[1]][1]))
```
For answers, there are 3 types of class, "status answered-accepted", "status unanswered" and "status answered":
```
answers = sapply(posts, function(x) xpathSApply(x, ".//div[@class = ('status answered-accepted')
or @class = ('status unanswered') or @class = ('status answered')]//strong", xmlValue))
```
For reputaion:
```
reputation = sapply(posts, function(x) xpathSApply(x, ".//span[@class = 'reputation-score']",
xmlValue))
```
For url ,we can only extract part of the actual url, so we have to paste it aftering extracting.

```
url_head = "http://stackoverflow.com"|
url = sapply(posts, function(x) xpathSApply(x, ".//a[@class = 'question-hyperlink']", function(y)
xmlGetAttr(y, "href")))
url_final = paste0(url_head, url)|
```

Then cbind them as a dataframe, and change the class of each row into the class it should be.

There is one more problem that we need to pay attention to, we may get NULL or list() if find no record to the corresponding node.
And we need to convert these NULL or list() to NA.

Q3. Obtain the URL for the "next" page listing posts

Design a function to return the url of next page, if there is no next page, we return NULL.

```
get_next_page = function(page){
  url_head = "http://stackoverflow.com"
  nextpage = xpathSApply(page, "//a[@rel = 'next']", function(x) xmlGetAttr(x, "href"))
  if(length(nextpage) == 0){
    nextpage
  }
  else{
    paste0(url_head, nextpage)
  }
}
```

Q4. repeat steps 1, 2, 3 and get all pages

Design a funciton `get_post = function(tag, n, url, i = 1, pre_data)` to get first "n" pages, if n is missing, we will get all pages.
iterative applying the method we use in Q1 and Q2, rbind the dataframe together and return it when we cannot find next page or the pages we get is equal to "n".
The detail of this function can be find at **code appendix**
display the first 6 rows

```
       qid                 date                       tags
1 34124599 2015-12-07 00:22:59Z  r; date; ggplot2; julian
2 34124573 2015-12-07 00:18:23Z          r; glm; bayesglm
3 34124493 2015-12-07 00:10:11Z       r; plot; regression
4 34124444 2015-12-07 00:05:17Z           r; class; apply
5 34124434 2015-12-07 00:04:25Z            r; conditional
6 34124319 2015-12-06 23:50:10Z r; if-statement; for-loop
                                                 title
1           R - plot julian day in x-axis using ggplot2
2  Comparing two glm models in for titanic dataset in R
3    how to get residuals plot in ridge regression in r
4 R -apply- convert many columns from numeric to factor
5            How to get R to check numbers? [duplicate]
6                              debug the if statement
                                                                                                  url_final
1            http://stackoverflow.com/questions/34124599/r-plot-julian-day-in-x-axis-using-ggplot2
2 http://stackoverflow.com/questions/34124573/comparing-two-glm-models-in-for-titanic-dataset-in-r
3   http://stackoverflow.com/questions/34124493/how-to-get-residuals-plot-in-ridge-regression-in-r
4  http://stackoverflow.com/questions/34124444/r-apply-convert-many-columns-from-numeric-to-factor
5                        http://stackoverflow.com/questions/34124434/how-to-get-r-to-check-numbers
6                              http://stackoverflow.com/questions/34124319/debug-the-if-statement
  views votes answers            user reputation
1     5     0       0    thiagoveloso        396
2     6     1       0          haimen        123
3     6     1       0         Leonard          6
4     9     1       2          GabyLP        676
5    12     0       0 Ozgur Alptekn          7
6     8     0       1       Boro Dega         28
```

After getting all 2331 pages, we use saveRDS to save the dataframe. `saveRDS(Allpages, file = "~/Allpages.rds")`

Part 2

Design a function `get_info_eachpost = function(url)` to return the question, answers, comments in a post.
Design two sub functions to complete the task.
The first function is designed to get the question information and answer information in a post, because the patterns of question and answers are very similar.
The second function is designed to get the comment, and we will seperate them as comments of question and comments of answers. because we need to return the "parent" of these comments, and if we do it seperatly, we can find their "parents" easily.
For question and answer:
Get the node of question and answer:

```
page_question = xpathSApply(page, "//div[@class = 'question']")
page_answer = xpathSApply(page, "//div[@class = 'answer accepted-answer' or @class = 'answer']")
```

Question and answers all have the same pattern for "user", "userid", "reputation", "score", "text",

```
user = xpathSApply(page, ".//div[@class = 'user-details']//a", xmlValue)
userid = xpathSApply(page, ".//div[@class = 'user-gravatar32']//a", function(x) gsub("/.*?/(.*?)/.*",
"\\1", xmlGetAttr(x, "href")))
reputation = xpathSApply(page, ".//span[@class = 'reputation-score']", xmlValue)
score = xpathSApply(page, ".//div[@class = 'vote']//span[@class = 'vote-count-post ']", xmlValue)
text = as(getNodeSet(page, ".//div[@class = 'post-text']")[[1]], "character")
```

For "date", our pattern is a little bit complicated, because there are edited time and asked time, we need first find the text that cotains "ask" or "answer" and then return to its parent to find "date".
For question:

```
date = xpathSApply(page, ".//td[@class = 'post-signature' or @class = 'post-signature owner']//
 text()[contains(., 'ask')]/..//span", function(x) gsub("([^A-z]*).*", "\\1", xmlGetAttr(x,
"title")))
id = xmlGetAttr(page, "data-questionid")
qid = id
```

For answer:

```
date = xpathSApply(page, ".//td[@class = 'post-signature' or @class = 'post-signature owner']//
 text()[contains(., 'answer')]/..//span", function(x) gsub("([^A-z]*).*", "\\1", xmlGetAttr(x,
"title")))
top_ancestor = xmlAncestors(page)[[1]]
qid = xpathSApply(page, "//div[@class = 'question']", function(x) xmlGetAttr(x, "data-questionid"))
```

Their "parent" are NA.

For comments:
Comments of question and comments of answers have the same patterns in ?user?, "userid", "date", "reputation", "score", "text", "id", "qid", their only difference is parent, one is qid and one is answer-id.

```
  user = xmlSApply(page, function(x) xpathSApply(x, ".//a[@class = 'comment-user' or
   @class = 'comment-user owner']", xmlValue))
  userid = xmlSApply(page, function(x) xpathSApply(x, ".//a[@class = 'comment-user' or
   @class = 'comment-user owner']/@href", function(x) gsub("/.*?/(.*?)/.*", "\\1", x)))
  userid = unname(userid)
  date = xmlSApply(page, function(x) xpathSApply(x, ".//span[@class = 'comment-date']//
   span/@title", function(x) gsub("([^A-z]*).*", "\\1", x)))
  date = unname(date)
```

```
  reputation = xmlSApply(page, function(x) xpathSApply(x, ".//a[@class = 'comment-user' or
   @class = 'comment-user owner']/@title", function(x) gsub(".*?([0-9]+).*", "\\1", x)))
  reputation = unname(reputation)
  score = xmlSApply(page, function(x) xpathSApply(x, ".//td[@class = ' comment-score']",
   function(x) gsub(".*?([0-9]*).*", "\\1", xmlValue(x))))
  text = xmlSApply(page, function(x) as(getNodeSet(x, ".//span[@class = 'comment-copy']")[[1]],
"character"))
  id = xmlSApply(page, function(x) gsub(".*?([0-9]+).*", "\\1", xmlGetAttr(x, "id")))
  top_ancestor = xmlAncestors(page_given)[[1]]
  qid = xpathSApply(top_ancestor, "//div[@class = 'question']", function(x)
   xmlGetAttr(x, "data-questionid"))
  qid = rep(qid, n)
```

For comments of questions, parent = qid, else:

```
parent = rep(xmlGetAttr(page_given, "data-answerid"), n)
```

The details of my function can be found in the **code appendix**

Randomly check some url to see if our function is correct.

```
> get_info_eachpost(test_url[1])
             user  userid                date reputation score
1        Vasily A 1707278 2014-06-10 15:50:34        760     1
2 G. Grothendieck  516548 2014-06-10 16:04:55      66780
3        Vasily A 1707278 2014-06-10 16:13:06        760
4 G. Grothendieck  516548 2014-06-10 16:14:56      66780
5        Vasily A 1707278 2014-06-10 16:20:42        760
6 G. Grothendieck  516548 2014-06-10 16:25:37      66780

text
1 <div class="post-text" itemprop="text">&#13;\n&#13;\n<p>In my code, I use the function which
does simple thing: takes records selected by given condition and adds give text to "errors"
column:</p>\n\n<pre><code>dterr &lt;- function (dtIn, condition, errtext) {\n  if
(!is.element('errors', names(dtIn))) {dtIn[,errors:=""];}\n  dtIn[eval(substitute(condition)),
errors:={paste0(errors, errtext)}];\n  invisible(dtIn);\n}\n</code></pre>\n\n<p><sup>   (I
am thankful to people who helped with my previous <a
href="http://stackoverflow.com/questions/24071896/setting-i-condition-passed-as-argument">question</a>
about <code>eval(substitute)</code>)</sup></p>\n\n<p>So, in simple cases the function works
as expected:</p>\n\n<pre><code>dt1 &lt;- fread(\n  "id,colA,colB\n   id1,3,xxx\n   id2,0,zzz\n
  id3,NA,yyy\n   id4,0,aaa\n     ")\n\nmyNum=0\n\ndterr(dt1, colA&gt;myNum, "positive");\ndt1\n#
   id   colA colB   errors\n# 1: id1    3  xxx   positive\n# 2: id2    0  zzz         \n# 3:
id3   NA  yyy         \n# 4: id4    0  aaa  \n</code></pre>\n\n<p>But when I try to call it
from another function, in combination with the variable defined inside, I get an error:</p>\n\n<pre>
<code>myfun &lt;- function(){\n  myNum2=1;\n  dterr(dt1, colA&gt;myNum2, "big!");\n}\n\nmyfun()\n#
Error in
eval(expr, envir, enclos) : object 'myNum2' not found \n</code></pre>\n\n<p>Obviously, I should
modify my function to pass the correct environment to <code>eval</code> - but for the moment
I can't find correct solution (I played with different combinations of
<code>parent.frame()</code>, <code>environment()</code>, etc). Any hints are appreciated.</p>\n
   </div>
2
<span class="comment-copy">Wrap the body of <code>dterr</code> in <code>eval.parent(substitute({
... }))</code> .</span>
3
<span class="comment-copy">wow, it works! thanks! Maybe you can add it as an answer, so I will
mark it accepted (I would be especially grateful if you can add just few words of explanation
as I am not sure to understand exactly what happens by this code).</span>
4
```

```
<span class="comment-copy">Surely this is not significantly different than the other question.
<code>substitute</code> substitutes the arguments into the code and then <code>eval.parent</code>
evaluates the code with the substitutions in the caller's environment.</span>
5
<span class="comment-copy">ok, I see. I was just confused that such construct leads to
<code>{eval.parent(substitute({ ... eval(substitute(...))}))}</code> which blew my mind :)</span>
6
<span class="comment-copy">You can remove the inner eval/substitute. That is
<code>eval(substitute(condition))</code> becomes just <code>condition</code>.</span>
        id     type   parent      qid
1 24145447 question     <NA> 24145447
2 37259696  comment 24145447 24145447
3 37260017  comment 24145447 24145447
4 37260093  comment 24145447 24145447
5 37260289  comment 24145447 24145447
6 37260473  comment 24145447 24145447
```

After checking carefully from the website, we find that our function works well.

Part 3

Q1. What is the distribution of the number of questions each person answered?
Split the data by user, and count the number of answers each person answered, to make our plot looks better, we will exclude the outlier.
We will get 2 plots, one includes users who have no contributions at all, and another one only with users who have made at least 1 contribution.

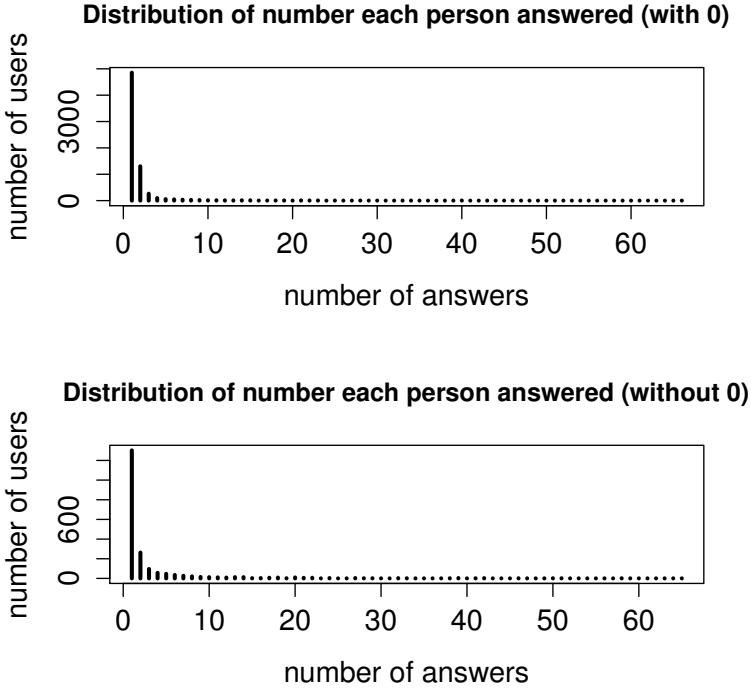


Figure 1: distribution plot

Q2. What are the most common tags?

Since there is not tags in "rQAs", we will use our result from Part 1, and merge it with "rQAs". In order to merge these 2 dataframes, we have the change the colname "id" of "Allpages" to "qid".

```
merge_result = merge(rQAs_new, Allpages[,c("qid","tags")], by = "qid")
```

Using "strsplit" and "unlist" to get all tags.

```
> merge_result_split = split(merge_result, merge_result$qid)
> all_tags = sapply(merge_result_split, function(x) unique(x$tags))
> all_tags = strsplit(all_tags, "; ")
> all_tags = unlist(all_tags)
> sort(table(all_tags), decreasing = TRUE)[1:10]

all_tags
         r    ggplot2       plot      shiny data.frame data.table      dplyr     matrix
      8784        732        409        401        381        304        304        206
   rstudio      regex
       189        166
```

Q3. How many questions are about ggplot?
Subset: get the rQAs whose "type" is question.
If we use tag to find questions about ggplot, then there are 732 quesitons about ggplot.
Now, if we use text to find quesitons about ggplot.
Find rQAs whose type is "question":

```
> rQAs_question = rQAs[rQAs$type == "question", ]
> table(grepl("ggplot", rQAs_question$text, ignore.case = TRUE))
FALSE  TRUE
 9045   959
```

So there are 959 questions with "ggplot" in text.

We can also define quesitons about "ggplot" as the combination of these two, its text includes "ggplot" or its tags include "ggplot".

```
> merge_result_question = merge_result[merge_result$type == "question", ]
> table(grepl("ggplot", merge_result_question$text, ignore.case = TRUE) | grepl("ggplot",
+ merge_result_question$tags, ignore.case = TRUE))

FALSE  TRUE
 9385  1010
```

Q4. How many questions involve XML, HTML or Web Scraping?
Using our subset rQAs_question's "tags" or "text" to match "xml—html—web-scrapping—(web scraping)"

```
> table(grepl("xml|html|web-scrapping|(web scrapping)", merge_result_question$tags, ignore.case
= TRUE))

FALSE  TRUE
 9122  1273
> table(grepl("xml|html|web-scrapping|(web scrapping)", merge_result_question$text, ignore.case
= TRUE))

FALSE  TRUE
 8600  1795
```

Because "text" always have some irrelevant information, so we would use the result of tags, there are approximately 1300 questions about xml, html, web-scrapping.

Q5. What are the names of the R functions referenced in the titles of the posts?

First, it is impossible for as to find all R functions, because there are so many packages, and we can only recognize functions in the packages that we required.

Choose some packages to library, from the result of **Q2**, we can find some most common tags, and we will only library those popular packages.
The packages we choose are "" Get the title of each quesiton, then we have 2 methods to find the R functions.

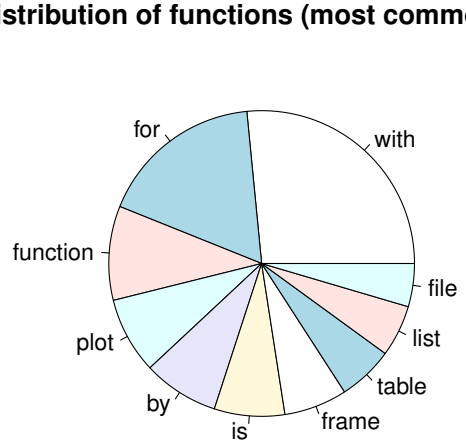
Method 1 split our title by space or "(", and check if each string is a function.
Design a function to check if a string is a function:

```
is_func = function(x){
  tryCatch(is.function(get(x)), error = function(e) FALSE)
}
```

Get our result.

```
> func_result = unlist(func_result)
> sort(table(func_result), decreasing = TRUE)[1:10]
func_result
    with      for function     plot       by       is    frame    table     list     file
    1450      949      544      441      439      407      365      320      298      249
```

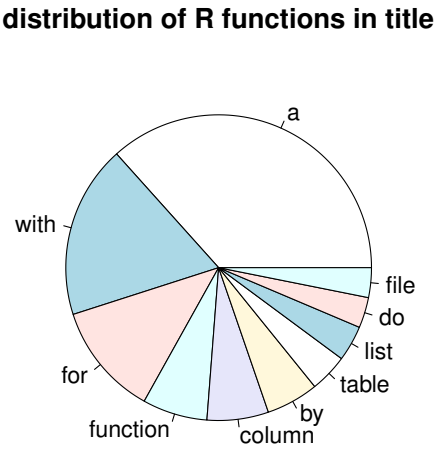
This result is not very good, apparently, not all "with", "for", "by".. are functions, they can be used as simple words. however it is hard for us to find the difference between function 'a" and word "a".



Method 2
Using ls() to find find all functions in our R packages, them iteratively find the times they appear for each function.
However in this method, we need to spend a lot of effort dealing with punctuations, such as "-", ":"
In order to transform the "functions" into the form of regular expression, we need to add "\\" before all

punctuations.
This procedure is complicated, code can be found at **code appendix**

```
> func_table = sort(func_list, decreasing = TRUE)[1:10]
> func_table
       a     with      for function   column       by    table     list       do     file
    2902     1450      947      544      515      439      320      298      256      249
```



Q6. What are the names of the R functions referenced in the accepted answers and comments?

It is hard for us to distinguish accepted answers and answers from rQAs, because we need to get the html of each question, and do the scrapping to find which answer is "accepted answer", however there are about 10000 unique "qid", which means we have to scrapping 10000 pages, it will cost a long time, about 4 times the time of Part 1.

So instead of using "accpeted answers" and comments, we will just use answers and comments.

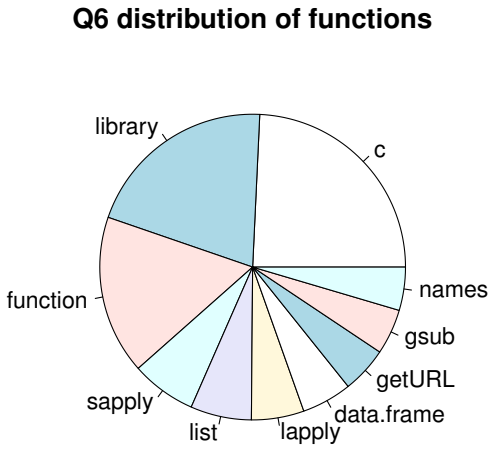
Using regular expression to get the "code" from "text". the pattern is like `"<code>...<\code>"`

```
> Allcodes = regmatches(rQAs_new_ans_com$text, gregexpr("<code>.*?</code>", rQAs_new_ans_com$text))
> Allcodes = lapply(Allcodes, function(x) gsub("<code>|</code>", "", x))
> Allcodes = unlist(Allcodes)
```

Then extract function from "Allcode", the pattern of function is like this "xxxx(", which end with a "(", x could be letter, number or some punctuations, in order to get these punctuations, we find the unique punctuations in my function pool (over 2000 functions from the packages specified above).
Add them to my regular expression, and get the result.

```
> rQAs_new_ans_com = rQAs[rQAs$type == "answer" | rQAs$type == "comment", ]
#Find all possible "punct" in R functions.
> punct = regmatches(Allfunctions, gregexpr("[[:punct:]]", Allfunctions))
> punct = unique(unlist(punct))
> punct = punct[punct != "("]
> pattern_punct = paste0(punct, collapse = "\\")
> pattern_punct = paste0("\\", pattern_punct)
```

```
> pattern = paste0("([A-z0-9", pattern_punct, "]+)\\(")
#Fina all functions that follow our pattern
> func_result = regmatches(Allcodes, gregexpr(pattern, Allcodes))
#Replace "(" with nothing
> func_result = unlist(func_result)
> func_result = gsub("\\(", "", func_result)
> table_func = head(sort(table(func_result), decreasing = T), 10)
> table_func
func_result
         c    library   function     sapply       list     lapply data.frame     getURL       gsub
      8748       7381       6062       2483       2327       2007       1914       1761       1732
> pie(table_func)
```



This result a lot better than the result from question 5, because the information from code line is more relevant to R functions.

Code Appendix

```
#**********************Assignment6**********************
library(RCurl)
library(XML)
library(utils)

options(expressions=500000)
#*******Part1*********
#Process the first page.
initial_url = "http://stackoverflow.com/questions/tagged/r?sort=newest"
page_content = getURLContent(initial_url)
class(page_content)
nchar(page_content)
cat(substring(page_content,1,2000))

page = htmlParse(page_content)
class(page)


#The pattern of posts
#? if I want to extract posts with class contains question?
posts = getNodeSet(page, "//div[@class = ('question-summary')]")
#Exactly 15 posts

p = posts[[1]]
p1 = getNodeSet(p, ".//div[@class = 'user-details']//a", xmlValue)
xpathSApply(p, ".//div[@class = 'user-details']//a", xmlValue)


get_post = function(tag, n, url, i = 1, pre_data){
  #url1 = "http://stackoverflow.com/questions/tagged/r?page=1662&sort=newest&pagesize=50"
  if(missing(url)){
    url = paste0("http://stackoverflow.com/questions/tagged/", tag, "?page=1&sort=newest&pagesize=50")
  }
  #page_content = getURLContent(url, .mapUnicode = FALSE)
  page_content = getURL(url, .mapUnicode = FALSE)
  page = htmlParse(page_content, asText = TRUE)
  posts = getNodeSet(page, "//div[@class = ('question-summary')]")
  #tags
  tags = sapply(posts, function(x) xpathSApply(x, ".//div//a[@class = 'post-tag']", xmlValue))
  tags = sapply(tags, function(x) paste(x, collapse = "; "))
  tags = unname(tags)
  #Note that we could also use date_posts = xpathSApply(page, ".//div//a[@class = 'post-tag']")
  #The reason I use sapply, is that I want to make sure the date and user are extracted from
the same post.
  #This thought is also applied in the following quesiton, so I will use sapply to our "posts"

  #id
  id = sapply(posts, function(x) gsub(".*?([0-9]+).*","\\1", xmlGetAttr(x, "id")))
  #user
  user = sapply(posts, function(x) xpathSApply(x, ".//div[@class = 'user-details']//a", xmlValue))
  #date
  date = sapply(posts, function(x) xpathSApply(x, ".//span[@class = 'relativetime']", function(y)
xmlGetAttr(y,"title")))
  #title
  title = sapply(posts, function(x) xpathSApply(x, ".//div[@class = 'summary']//a[@class =
```

```
'question-hyperlink']", xmlValue))
  #votes
  votes = sapply(posts, function(x) xpathSApply(x, ".//div[@class = 'votes']//strong", xmlValue))
  #view
  views = sapply(posts, function(x) xpathSApply(x, ".//div[@class = 'views ' or @class = 'views
supernova' or @class = 'views warm' or @class = 'views hot']", function(x) strsplit(xmlGetAttr(x,
"title"), " ")[[1]][1]))
  #answer
  answers = sapply(posts, function(x) xpathSApply(x, ".//div[@class = ('status answered-accepted')
or @class = ('status unanswered') or @class = ('status answered')]//strong", xmlValue))
  #reputation
  reputation = sapply(posts, function(x) xpathSApply(x, ".//span[@class = 'reputation-score']",
xmlValue))
  #url
  url_head = "http://stackoverflow.com"
  url = sapply(posts, function(x) xpathSApply(x, ".//a[@class = 'question-hyperlink']", function(y)
xmlGetAttr(y, "href")))
  url_final = paste0(url_head, url)
  #nest page
  next_url = get_next_page(page)
  data = data.frame(cbind(id, date, tags, title, url_final, views, votes, answers, user, reputation))
  data[] = lapply(data, as.character)
  #Release the memory, which can make my computation faster especially to the pages after 1000.
  rm(list = c("page_content", "page", "posts", "tags", "id", "user", "date", "title", "votes",
"views", "answers", "reputation", "url_head", "url", "url_final"))
  if(!missing(pre_data)){
    data = rbind(pre_data, data)
  }

  i = i + 1
  print(i)
  if(missing(n)){
    if(length(next_url) == 0){
      return(data)
    }
    else{
      get_post(tag = tag, url = next_url, i = i, pre_data = data)
    }
  }
  else{
    if(i > n | length(next_url) == 0){
      return(data)
    }
    else{
      get_post(tag, n, next_url, i, data)
    }
  }
}

get_next_page = function(page){
  url_head = "http://stackoverflow.com"
  nextpage = xpathSApply(page, "//a[@rel = 'next']", function(x) xmlGetAttr(x, "href"))
  if(length(nextpage) == 0){
    nextpage
  }
  else{
```

```
    paste0(url_head, nextpage)
  }
}

#all_post_500pages = get_post(tag = "r", n = 500)
#result = get_post("r")
#save(all_post_500pages, file = "~/all500pages.csv")
#write.csv(all_post_500pages, file = "~/all500pages.csv")
#all500pages = read.csv(file = "~/all500pages.csv")

#write.csv(result, file = "~/Allpages.csv")
#Allpages = read.csv(file = "~/Allpages.csv")
Allpages = Allpages[,-1]
Allpages = get_post("r")
Allpages = Allpages[!duplicated(Allpages$qid),]
saveRDS(Allpages, file = "~/Allpages.rds")
loadRDS("~/Allpages.rds")



#********Part2************
load("~/academic/rQAs.rda")


get_info_eachpost = function(url){
  page_content = getURL(url, .mapUnicode = FALSE)
  page = htmlParse(page_content, asText = TRUE)
  page_question = xpathSApply(page, "//div[@class = 'question']")
  page_answer = xpathSApply(page, "//div[@class = 'answer accepted-answer' or @class = 'answer']")
  data_question = lapply(page_question, function(x) get_data_eachtype("question", x))
  data_answer = lapply(page_answer, function(x) get_data_eachtype("answer", x))
  data_comment_Q = lapply(page_question, function(x) get_comment("comment_Q", x))
  data_comment_A = lapply(page_answer, function(x) get_comment("comment_A", x))
  data_answer_full = do.call(rbind, data_answer)
  data_comment_Q_full = do.call(rbind, data_comment_Q)
  data_comment_A_full = do.call(rbind, data_comment_A)
  data = list(data_question[[1]], data_answer_full, data_comment_Q_full, data_comment_A_full)
  return(do.call(rbind, data))
  }


get_data_eachtype = function(type_given, page){
    user = xpathSApply(page, ".//div[@class = 'user-details']//a", xmlValue)
    userid = xpathSApply(page, ".//div[@class = 'user-gravatar32']//a", function(x)
      gsub("/.*?/(.*?)/.*", "\\1", xmlGetAttr(x, "href")))
    reputation = xpathSApply(page, ".//span[@class = 'reputation-score']", xmlValue)
    score = xpathSApply(page, ".//div[@class = 'vote']//span[@class = 'vote-count-post ']",
xmlValue)
    text = as(getNodeSet(page, ".//div[@class = 'post-text']")[[1]], "character")
    if(type_given == "question"){
      date = xpathSApply(page, ".//td[@class = 'post-signature' or @class = 'post-signature
owner']//
        text()[contains(., 'ask')]/..//span", function(x) gsub("([^A-z]*).*", "\\1", xmlGetAttr(x,
"title")))
      id = xmlGetAttr(page, "data-questionid")
      qid = id
```

```
      parent = NA
    }
    else{
      date = xpathSApply(page, ".//td[@class = 'post-signature' or @class = 'post-signature
owner']//
        text()[contains(., 'answer')]/..//span", function(x) gsub("([^A-z]*).*", "\\1", xmlGetAttr(x,
"title")))
      id = xmlGetAttr(page, "data-answerid")
      parent = NA
      top_ancestor = xmlAncestors(page)[[1]]
      qid = xpathSApply(page, "//div[@class = 'question']", function(x)
       xmlGetAttr(x, "data-questionid"))
    }
    type = type_given
    data = data.frame(cbind(user, userid, date, reputation, score, text, id, type, parent,
qid))
    data[] = lapply(data, as.character)
    return(data)
}


get_comment = function(type_given, page_given){
  page = getNodeSet(page_given, ".//tr[@class = 'comment ']")
  n = length(page)
  user = xmlSApply(page, function(x) xpathSApply(x, ".//a[@class = 'comment-user' or
   @class = 'comment-user owner']", xmlValue))
  userid = xmlSApply(page, function(x) xpathSApply(x, ".//a[@class = 'comment-user' or
   @class = 'comment-user owner']/@href", function(x) gsub("/.*?/(.*?)/.*", "\\1", x)))
  userid = unname(userid)
  date = xmlSApply(page, function(x) xpathSApply(x, ".//span[@class = 'comment-date']//
   span/@title", function(x) gsub("([^A-z]*).*", "\\1", x)))
  date = unname(date)
  reputation = xmlSApply(page, function(x) xpathSApply(x, ".//a[@class = 'comment-user' or
   @class = 'comment-user owner']/@title", function(x) gsub(".*?([0-9]+).*", "\\1", x)))
  reputation = unname(reputation)
  score = xmlSApply(page, function(x) xpathSApply(x, ".//td[@class = ' comment-score']",
   function(x) gsub(".*?([0-9]*).*", "\\1", xmlValue(x))))
  text = xmlSApply(page, function(x) as(getNodeSet(x, ".//span[@class = 'comment-copy']")[[1]],
"character"))
  id = xmlSApply(page, function(x) gsub(".*?([0-9]+).*", "\\1", xmlGetAttr(x, "id")))
  top_ancestor = xmlAncestors(page_given)[[1]]
  qid = xpathSApply(top_ancestor, "//div[@class = 'question']",
   function(x) xmlGetAttr(x, "data-questionid"))
  qid = rep(qid, n)

  if(type_given == "comment_Q"){
    parent = qid
  }
  else{
    parent = rep(xmlGetAttr(page_given, "data-answerid"), n)
  }
  type = rep("comment", n)
  data = data.frame(cbind(user, userid, date, reputation, score, text, id, parent, type, qid))
  data[] = lapply(data, as.character)
  return(data)
}
```

```
#ramdonly test some urls
set.seed(10)
test_url = Allpages$url_final[sample(nrow(Allpages))[1:5]]

get_info_eachpost(test_url[1])


#****************Part3*****************
load("~/rQAs.rda")
dim(rQAs)
#(1)
#Use "userid" to distinguish different persons
#If a userid answered the same question for many times, we will only count it as 1 time.
par(mfrow = c(2,1), cex.main = 0.9)
rQAs_split_user = split(rQAs, rQAs$userid)
#rQAs_split_type = split(rQAs, rQAs$type)
user_answer = lapply(rQAs_split_user, function(x) x[x$type == "answer", ])
user_answer_unqiue = sapply(user_answer, function(x) length(unique(x$qid)))

table_answer = table(user_answer_unqiue)
#To make our plot looks better, we need to exclude the largest number of answer.
table_answer = table_answer[-length(table_answer)]
plot(table_answer, type = 'h', lwd = 3, main = "Distribution of number each person answered
(with 0)", xlab = "number of answers", ylab = "number of users")
length(unique(rQAs$user))



#If we want to find the distribution without those who has 0 answers
rQAs_answer = rQAs[rQAs$type == "answer", ]
rQAs_answer_split = split(rQAs_answer, rQAs_answer$userid)
user_answer_unqiue = sapply(rQAs_answer_split, function(x) length(unique(x$qid)))

table_answer = table(user_answer_unqiue)
table_answer = table_answer[-length(table_answer)]
plot(table_answer, type = 'h', lwd = 3, main = "Distribution of number each person answered
(without 0)", xlab = "number of answers", ylab = "number of users")



#(2)
#merge
rQAs_new = rQAs
rQAs_new$url_final = rownames(rQAs_new)
name = names(Allpages)
name[which(name == "id")] = "qid"
colnames(Allpages) = name
merge_result = merge(rQAs_new, Allpages[,c("qid","tags")], by = "qid")
names(merge_result)
#Randomly check if we get the correct dataframe.
#All correct
merge_result$tags = as.character(merge_result$tags)
#Since there are many answers and comments in one question, so we need to use qid to split
'merge_result'
merge_result_split = split(merge_result, merge_result$qid)
all_tags = sapply(merge_result_split, function(x) unique(x$tags))
```

```
all_tags = strsplit(all_tags, "; ")
all_tags = unlist(all_tags)
sort(table(all_tags), decreasing = TRUE)[1:10]
pie(sort(table(all_tags), decreasing = TRUE)[1:10])

#(3)
#If we use tag to find questions about ggplot, then there are 732 quesitons about ggplot
#Now, if we use text to find quesitons about ggplot.
#Find rQAs whose type is "question"
rQAs_question = rQAs[rQAs$type == "question", ]
table(grepl("ggplot", rQAs_question$text, ignore.case = TRUE))
#So there are 959 questions with "ggplot" in text.

#We can also define quesitons about "ggplot" as the combination of these two, its text includes
"ggplot" or its tags include "ggplot".
merge_result_question = merge_result[merge_result$type == "question", ]
table(grepl("ggplot", merge_result_question$text, ignore.case = TRUE) |
grepl("ggplot", merge_result_question$tags, ignore.case = TRUE))
#Then there are 1009 qualified quesitons.



#Use title to define questions about "ggplot".
rQAs_new_question = rQAs_new[rQAs_new$type == "question", ]
table(grepl("ggplot", rQAs_new_question$url_final, ignore.case = TRUE))
#So there are 516 questions about


#(4)
#Using tags to find them.
table(grepl("xml|html|web-scrapping|(web scrapping)", merge_result_question$tags, ignore.case
= TRUE))
table(grepl("xml|html|web-scrapping|(web scrapping)", merge_result_question$text, ignore.case
= TRUE))
#There are 1273 questions about


#(5)
url = rQAs_new_question$url_final

Alltitles = gsub("http://stackoverflow.com/questions/([0-9]+)/", "", url)
title_str = strsplit(Alltitles, "-| ")
title_str = unlist(title_str)

#Method 1
is_func = function(x){
  tryCatch(is.function(get(x)), error = function(e) FALSE)
}

func_result = title_str[sapply(title_str, is_func)]
#func_result = unlist(func_result)
sort(table(func_result), decreasing = TRUE)[1:10]
pie(sort(table(func_result), decreasing = TRUE)[1:10], main = "distribution of functions (most
common 10)")
```

```
#Method 2
title_str_all = paste(title_str_all, collapse = " ")

#However it is impossible for us to find all functions in R, because there are so many packages,
and we cannot install all of them.
#Use the result of question2 "most common tags" to find our target packages and install them.
#They are "base" "ggplot"
#And I am going to recognize the function
#install.packages("ggplot2")
#install.packages("shiny")
#install.packages("dplyr")
#install.packages("knitr")
#install.packages("stringr")
#install.packages("stringi")
target_packages = list("base", "ggplot2", "shiny", "dplyr", "knitr", "stringr", "stringi")
invisible(
  lapply(target_packages, require, character.only = TRUE)
)

target_packages_name = paste0("package:", target_packages)
Allfunctions = lapply(target_packages_name, ls)
Allfunctions = unlist(Allfunctions)


#functions_in_title = Allfunctions[Allfunctions %in% tolower(title_str_all)]
#Method 1
punct = regmatches(Allfunctions, gregexpr("[[:punct:]]", Allfunctions))
punct = unique(unlist(punct))
punct = punct[punct != ">" & punct != "<"]
punct_reg = paste0("\\", punct)
punct_rep = paste0("\\\\", punct)

Allfunctions_new = Allfunctions
for(i in 1:length(punct_reg)){
  Allfunctions_new = gsub(punct_reg[i], punct_rep[i], Allfunctions_new)
  print(i)
  print(head(Allfunctions_new))
}


functions_in_title = lapply(Allfunctions_new, function(x) length(regmatches(title_str_all,
gregexpr(x, title_str_all))[[1]]))
names(functions_in_title) = Allfunctions_new
func_list = unlist(functions_in_title)
sort(func_list, decreasing = TRUE)[1:10]
#This result is not good, because the most common functions we find are "t", "a", "n", "c",
"p"
#We need to change the pattern of our Allfunctions.
#The define our function as " function[ \\()]"
Allfunctions_pattern = paste0(" ", Allfunctions_new, "[ \\(]")
#And do the previous procedure again
functions_in_title = lapply(Allfunctions_pattern, function(x) length(regmatches(title_str_all,
gregexpr(x, title_str_all))[[1]]))
names(functions_in_title) = Allfunctions
func_list = unlist(functions_in_title)
```

```
func_table = sort(func_list, decreasing = TRUE)[1:10]
func_table
#this time we get a better result, but far from perfect, "a" is the most common one, and of
course it shouldn't, but now it is different to distinguish function "a" and the word "a".
#So my final result is this:
pie(func_table, main = "distribution of R functions in title")

#Note we can also use is.function() , get() and try() to find our functions, should get approximately
the same result


#(6)
#??Accepted answer
unique(rQAs$type)
rQAs_new_ans_com = rQAs[rQAs$type == "answer" | rQAs$type == "comment", ]
rQAs_new_ans_com$text[1]

Allcodes = regmatches(rQAs_new_ans_com$text, gregexpr("<code>.*?</code>", rQAs_new_ans_com$text))
Allcodes = lapply(Allcodes, function(x) gsub("<code>|</code>", "", x))
Allcodes = unlist(Allcodes)


unique(rQAs$type)
rQAs_new_ans_com = rQAs[rQAs$type == "answer" | rQAs$type == "comment", ]
#Find all possible "punct" in R functions.
punct = regmatches(Allfunctions, gregexpr("[[:punct:]]", Allfunctions))
punct = unique(unlist(punct))
punct
#The pattern of function is like this "XXXX(...)" X is letter, number or the "punct" I have
just find.
#However we have to remove "(" and from our pattern, because it can be confused with function"(").
punct = punct[punct != "("]
pattern_punct = paste0(punct, collapse = "\\")
pattern_punct = paste0("\\", pattern_punct)

pattern = paste0("([A-z0-9", pattern_punct, "]+)\\(")
#Fina all functions that follow our pattern
func_result = regmatches(Allcodes, gregexpr(pattern, Allcodes))
#Replace "(" with nothing
func_result = unlist(func_result)
func_result = gsub("\\(", "", func_result)
table_func = head(sort(table(func_result), decreasing = T), 10)
table_func
pie(table_func, main = "Q6 distribution of functions")
#We get a better result
```