Assignment 4

Filip Wilhelm Sjostrand

2023-06-02

```
# Packages -----------
library(httr)
library(rvest)
library(xml2)
library(purrr)
library(lubridate)
library(dplyr)
source("functions.R")
```

1 Getting all the questions

First, we shall create a helper function to parse a page. Given a page URL, this function will download the page content and parse it into an HTML document that we can extract data from. In the process it will check if the request was successful.

```
first_page_url <- "https://stackoverflow.com/questions/tagged/r?tab=newest&pagesize=50"
first_page <- parse_page(first_page_url) # No error
```

Further, we have to create a function to extract links from the HTML. From the parsed page, our function will extract the URLs of the the assigned type. By inspecting the HTML using the browser, we find that question links are contained within `a` tags with a class of `s-link` and that pagination structure are contained within `a` tags with a class of `s-pagination--item js-pagination-item`. The function will return all links of such classes.

Below, using `length(question_links)` and `head(question_links)` we can confirm that the question links are 50 and contains proper links. Printing `page_links` confirms its correctness too.

```
question_link <- get_links(first_page, "//a[@class='s-link']")
page_links <- get_links(first_page, "//a[@class='s-pagination--item js-pagination-item']")
```

Since we are interested in the first three pages and the last page we shall select those from the `page_links`. We can see that, simply, the first three links are the first three pages. However, the last link is not the last page as it is the "next" button, therefore we have to select the second to last link. Visually confirms that it is correct.

```
selected_pages <- c(first_page_url, page_links[1:2], page_links[5])
```

To get links of all of the question, we combine our functions. However, we do not want make too many request, thus utilize `slowly()` to add a 1 second delay to each function call. We can confirm with `length(question_pages)` that there are 200 question links in the vector.

```
parse_page_slowly <- purrr::slowly(parse_page, rate = rate_delay(1))
pages <- lapply(selected_pages, parse_page_slowly)
question_links <- unlist(lapply(pages, get_links, "//a[@class='s-link']"))
question_pages <- lapply(question_links, parse_page_slowly)
```

2 Scraping data

2.1 Posts

Now lets start scraping question data. Given a question URL, to simplify things, we will create separate functions to extract each piece of information. To keep data tidy, we shall create separate tables with similar characteristics. Beginning with "post data".

Let's start with extracting the number of views and votes for a question. Upon inspecting the HTML structure of a Stack Overflow question page, you'll see that the number of views is contained in a `div` tag with class `flex--item ws-nowrap mb8 mr16`, while the number of votes is contained in a span tag with class vote-count-post.

We further realize that most of the data is available in a similar way. To remove redundancy in the writing, we simply follow the same procedure to allocate the data to be scraped. That is, we inspect the desired using the browser. Further, we use xpath to allocate said data. Lastly, we return the data in a relevant data format.

We start with the first page and could confirm a successful extraction by comparing with its URL visually in the browser. We iterate over all pages with a for loop to confirm they will all have the same number of observations (6) as the first. When an error was thrown we investigated. For instance, `question_link[[144]]` is a "migrated" question to Stack Exchange. Hence, our functions were not fitted for this. We found that the view count and user name html elements differed, and therefore adjusted the functions accordingly. by `length(ll) = 200`,`unique(nr) = 6`, and `unique(ll) = 6`, the parsing appears correct. In other words, we have 200 questions scraped, all have 6 data observations, with no more than one row per question.

```
ll <- c()
nr <- c()
for (i in 1:length(question_pages)) {
    pp <- question_pages[[i]]
    df_post <- post_row(pp)
    ll[i] <- length(df_post)
    nr[i] <- nrow(df_post)

}
```

Simply combining all observations into a data.frame. Since the questions are handled in the order we've parsed the pages, we'll let it set the question IDs.

```
posts_data <- lapply(question_pages, post_row)
posts_df <- do.call(rbind, posts_data)
posts_df$index <- 1:nrow(posts_df)
```

2.2 Users

Further scraping refers to data that concerns the user who posted the question. Similar approach to as above, however, each user does not have all three types of badges. Thus, the function has adaptability to varying

amounts. Using the same confirmation techniques as above, it appears as if we have successfully scraped each question of user data.

```
ll <- c()
nr <- c()
for (i in 1:length(question_pages)) {
    pp <- question_pages[[i]]
    df <- get_user_details(pp)
    ll[i] <- length(df)
    nr[i] <- nrow(df)
}
```

Creating our user data table. Similar as before, we let the parsing order determine the question ID.

```
users_data <- lapply(question_pages, get_user_details)
users_df <- do.call(rbind, users_data)
users_df$index <- 1:nrow(users_df)
```

2.3 Editors

Edits are find in "https://stackoverflow.com/posts/.../revisions". Therefore we need to get these sites as well. We allocate them under the `user-action-time` class. We have modified `get_links()` to return `NA` if no `href` attribute is found. To later refer back to the correct question ID, we append it now. We have to remove `NA`s for `parse_page_slowly()` to be able to parse the links.

```
revision_links <- unlist(lapply(question_pages, get_links, "//div[@class = 'post-signature flex--item'],

link_df <- data.frame(revision_links) %>%
    mutate(index = 1:length(revision_links))

revision_links <-  df <- na.omit(link_df) # Remove NAs
revisions <- lapply(revision_links$revision_links, parse_page_slowly)
```

without explicitly writing it out. We used the same technique as before to make sure consistency in the scraping of data.

```
ll <- c()
nr <- c()
for (i in 1:length(revisions)) {
    pp <- revisions[[i]]
    df <- editors_times(pp)
    ll[i] <- length(df)
    nr[i] <- nrow(df)
}
```

Now we can merge all of the rows but also refer back to the indeces we created before.

```
edit_data <- lapply(revisions, editors_times)
for (i in 1:length(edit_data)) {
    edit_data[[i]] <- edit_data[[i]] %>%
        cbind(index = revision_links$index[[i]])
}
edit_df <- do.call(rbind, edit_data)
```

2.4 Answers

For this, we can reuse the pages we've parsed in `question_pages`. We shall utilize the same indexing technique as for the revisions. The approach is to first isolate each indivudal question. Then, for each question we scrape releveant data, utilizing previous methods and functions to the createst extent.

```
qlinks_index <- data.frame(question_links) %>%
    mutate(index = 1:length(question_links))

answer_df <- list()
for (i in 1:length(question_pages)) {
    page <- question_pages[[i]]
    answer_df[[i]] <- answer_per_question(page) %>%
        cbind(index = qlinks_index$index[[i]])
}

only_df <- answer_df[sapply(answer_df, is.data.frame)] # remove all NAs

answers_df <- do.call(rbind, only_df)
```

3 Final result

Below we've printed the head of all the tables. It is possible to merge all using the index.

```
# Function to limit string length to 20 characters
limit_string_length <- function(x) {
    if (is.character(x)) {
        ifelse(nchar(x) > 20, paste0(substr(x, 1, 20), "..."), x)
    } else {
        x
    }
}
```

Table 1: Posts Data

views	votes	body	tags	post_time	user	index
8	0	string <- "this is a...	r, string, substr	2023-06-03 03:00:13	Adrian	1
13	0	I have a data frame ...	r	2023-06-03 02:49:56	Aztec22	2
5	0	I'd like to simulate...	r, spatial, covarian...	2023-06-03 02:42:40	EM823823	3
9	0	I am using the GVIF ...	r, linear-regression...	2023-06-03 01:37:18	José Amorim	4
19	0	I am fairly new to u...	r, categorical-data,...	2023-06-03 00:06:53	AllyDavidge	5
20	0	I am new to Bayesian...	r, statistics, bayes...	2023-06-02 23:40:49	0yl3r	6

Table 2: User Data

user	reputation	gold	silver	bronze	index
Adrian	9257	24	73	131	1
Aztec22	31	0	0	7	2
EM823823	121	0	0	5	3
José Amorim	1	0	0	0	4
AllyDavidge	1	0	0	0	5
0yl3r	1	0	0	0	6

Table 3: Editor Data

editors	times	index
Adrian	2023-06-03 03:11:24	1
Adrian	2023-06-03 03:00:13	1
Phil	2023-06-03 03:13:45	2
Aztec22	2023-06-03 02:49:56	2
Jon Spring	2023-06-03 00:28:59	5
AllyDavidge	2023-06-03 00:06:53	5

Table 4: Answer Data

user	reputation	Gold	Silver	bronze	body	index
benson23	15200	8	18	38	The output of substr…	1
jpsmith	9359	5	14	33	You can use tidyr's …	2
Ricardo Semião e Cas…	3958	1	8	27	Would it be as simpl…	8
Jason Connelly	1	0	0	1	I was able to genera…	10
Onyambu	64500	3	23	53	To view the results …	18
Kat	14700	3	18	51	I haven't found a wa…	20