DA5020 - Week 8 Assignment Web Scraping Programaically

2019-09-21

In this week's assignment, we continue our pursuit for good burgers in specific neighborhoods in the Boston area using data from the yelp website. We will programatically extract specific fields in the data using the rvest package. This assignment will also provide practice in writing functions and loops. Some questions require you to complete code within a partially written function given a specification. Other questions will not provide starter code.

Questions

1. (20 points) Retrieve the contents of the first webpage for the yelp search as specified in Assignment 7 and write R statements to answer the following questions on the retrieved contents:

- How many nodes are direct descendents of the HTML `<body>` element (the actual visible content of a web page)?
- What are the nodes names of the direct descendents of the `<body>`?
- How many of these direct descendents have an `id` attribute?
- What is the css selector to select restaurants that are advertisements? (You may not see the ads if you are logged in on Yelp or have an ad blocker running.)

Here's some code to help you get started:

```
library(rvest)

## Warning: package 'rvest' was built under R version 3.4.3

page <- read_html("https://www.yelp.com/search?find_desc=burgers&start=0&l=Bo
ston,MA")

# list the children of the <html> element (the whole page)
html_children(page)

## {xml_nodeset (2)}
## [1] <head>\n<script>\n            (function() {\n                var mai .
..
## [2] <body id="yelp_main_body" class="jquery country-us logged-out react- .
..

# get the root of the actual html body
root <- html_node(page, 'body')
```

2. (50 points) Modify following parameterized function `get_yelp_sr_one_page` to extract a list of businesses on Yelp, for a specific search keyword, a specific location and a specific page of results. get_yelp_sr_one_page <- function(keyword, loc="Boston,

MA") { # Scrape Yelp's search results page for a list of businesses # Args: # keyword - the keyword for a search query, the "&find_desc=" parameter # loc - the location to search for, the "&find_loc=" parameter in the url # Return: # A data frame containing burger restaurant contents in one search # results.

parameterize the search results URL yelp_url <- 'https://www.yelp.com/search?find_desc=%s&find_loc=%s' # `sprintf` replace "%s" with positional arguments following the string # `URLencode` ensures blank spaces in the keywords and location are # properly encoded, so that yelp will be able to recognize the URL yelp_url <- sprintf(yelp_url, URLencode(keyword), URLencode(loc))

yelpsr <- read_html(yelp_url)

`html_nodes` allow us to extract pieces from an html document using # XPath or css selectors. Most of the time, you would only use css selectors # since they are much easier to interpret.

Here we use `.regular_search-result` to exclude ad entries # since ad entries do not have this class. # We single out the items first so we can safely use simpler selectors # to extract information inside each item items <- yelpsr %>% html_nodes("li.regular-search-result")

links <- items %>% html_nodes("a.biz-name") # trim=T (trim = True) removes whitespaces between html text names <- links %>% html_text(trim=T) urls <- links %>% html_attr("href") %>% # cleanup useless url parameters (which are used # by yelp for analytical tracking purpose) str_replace("\?osq=.*","") pricelevels <- items %>% html_nodes(".business-attribute.price-range") %>% html_text(trim=T) %>% str_count()

some results might not have neighborhood or address information, # therefore we cannot use selectors to select them into vectors directly. # (you will see a "vector length mismatch" error)

we must collect data item by item for these variables, i.e., go # from a column-wise approach to row-wise.

secondary_attrs <- items %>% #

is the parent of the neighborhood and address # attributes. it is unlikely that an item does not have this section at all. html_nodes('.secondary-attributes') %>% # pass each node in the selection to a `map` function purrr::map(function(item) { # collect those secondary attributes one by one # if an attribute is missing, it will be recorded as NA. tibble(neighborhood = item %>% html_node('.neighborhood-str-list') %>% html_text(trim=T), address = item %>% html_node('address') %>% html_text(trim=T), phone = item %>% html_node('.biz-phone') %>% html_text(trim=T)) }) %>% # merge rows bind_rows()

return a data frame tibble(name = names, url = urls, price = pricelevels # <—- Insert here more variables ——>) %>% cbind(secondary_attrs) }

example output

get_yelp_sr_one_page("burgers") %>% head(5) %>% select(name, neighborhood, address) %>% knitr::kable() ```

Add a parameter to the `get_yelp_sr_one_page` function so that it can scrape other pages other than the first page. E.g., `get_yelp_sr_one_page("burgers", page=2)` should return the results in the second page.

The modified function should return a data frame that contains the following information:

- restuarant name
- url to the yelp page of the restaurant
- price level
- service categories
- telphone number
- restuarant's neighborhood name, street address, city, state, and zipcode, all in separate columns
- average rating
- number of reviews
- URL to the restaurant's reviews list

3. (20 points) Write a function that reads multiple pages of the search results of any search keyword and location from Yelp.

Note that for some queries, Yelp may get a different number of results per page. You would need to either change the way you calculate the URL parameter, or use the `distinct(df)` function to remove duplicate rows.

4. (10 points) Optimize your function in question 3, add a small wait time (0.5s for example) between each request, so that you don't get banned by Yelp for abusing their website (hint: use `Sys.sleep()`).

Submission

You need to submit an .Rmd extension file as well as the generated pdf file. Be sure to state all the assumptions and give explanations as comments in the .Rmd file wherever needed to help us assess your submission. Remember to use the naming convention you have been following in this course so far.