Milad Tatari, DA5020 - Week 8 Assignment Web Scraping Programaically

11/02/2019

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: 1.1 How many nodes are direct descendents of the HTML `<body>` element (the actual visible content of a web page)?

```
library(rvest)

page <- read_html("https://www.yelp.com/search?find_desc=burgers&find_loc=Boston%2C%20MA&l=p%3AMA%3ABoston%3
# 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')
# direct descendent nodes of the HTML `<body>` element
DD.Nodes <- html_children(root)
#Print the results
print(paste("Number of Direct Descendent Nodes = ", length(DD.Nodes), sep=""))
```

```
## [1] "Number of Direct Descendent Nodes = 39"
```

1.2 What are the nodes names of the direct descendents of the `<body>`?

```
#let's use a for loop and print the node names
for (i in 1:39) {
  print(DD.Nodes[[i]])

}
```

```
## {html_node}
## <script>
## {html_node}
## <noscript>
## [1] <link rel="stylesheet" href="https://s3-media3.fl.yelpcdn.com/assets ...
## {html_node}
## <script id="yelp-js-error-reporting-init-error-reporting" type="application/json">
## {html_node}
## <script>
## {html_node}
## <div id="fb-root">
## {html_node}
## <script src="https://s3-media3.fl.yelpcdn.com/assets/2/www/js/ff37b3e6a199/assets/vendor_external/perimet
## {html_node}
## <noscript>
## [1] <div style="position:fixed; top:0; left:0; display:none" width="1" h ...
## {html_node}
## <div id="wrap" class="lang-en">
## [1] <div class="page-header">\n\n\n                          \n\n          ...
## [2] <span id="page-content" class="offscreen"> </span>
## [3] <div id="alert-container">\n\n    </div>
## [4] <div data-hypernova-key="yelp_main__9a5975765dfe75eb2053999c535d0df1 ...
## [5] <script type="application/json" data-hypernova-key="yelp_main__9a597 ...
## [6] <script>\n            (function() {\n                var main = null ...
## [7] <div style="display: none;" id="page-popups">\n            </div>
## {html_node}
## <script>
## {html_node}
## <noscript>
## [1] <img src="https://sb.scorecardresearch.com/p?cj=1&amp;c15=&amp;c3=&a ...
## {html_node}
## <script>
## {html_node}
## <noscript>
## [1] <img style="display: none;" src="https://pixel.quantserve.com/pixel/ ...
## {html_node}
## <script id="__LOADABLE_REQUIRED_CHUNKS__" type="application/json">
## {html_node}
## <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-polyfill/6.23.0/polyfill.min.js" crossorigin="a
## {html_node}
## <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/1.8.3/jquery.min.js">
## {html_node}
## <script>
## {html_node}
## <script src="https://cdnjs.cloudflare.com/ajax/libs/react/16.8.6/umd/react.production.min.js" crossorigin
## {html_node}
## <script src="https://cdnjs.cloudflare.com/ajax/libs/react-dom/16.8.6/umd/react-dom.production.min.js" cro
## {html_node}
## <script src="https://s3-media0.fl.yelpcdn.com/assets/public/module_yelp_main_webpack_runtime.yji-cacaa26c
## {html_node}
## <script src="https://s3-media0.fl.yelpcdn.com/assets/public/module_yelp_js_packages.yji-07273d822085248b
## {html_node}
## <script src="https://s3-media0.fl.yelpcdn.com/assets/public/common_chunk.yji-4c8a421b0526a319b4c1.chunk.
## {html_node}
## <script src="https://s3-media0.fl.yelpcdn.com/assets/public/module_core.yji-89a77722f870475f38f7.chunk.js
## {html_node}
## <script src="https://s3-media0.fl.yelpcdn.com/assets/public/module_search_results.yji-cf7d6c39d75c6831ce
```

```
## {html_node}
## <script>
## {html_node}
## <script src="//sb.scorecardresearch.com/beacon.js">
## {html_node}
## <script>
## {html_node}
## <script>
## {html_node}
## <script>
## {html_node}
## <script>
## {html_node}
## <script>
## {html_node}
## <script>
## {html_node}
## <noscript>
## [1] <img src="//secure-us.imrworldwide.com/cgi-bin/m?ci=us-804377h&amp;c ...
## {html_node}
## <script>
## {html_node}
## <div id="ttdUniversalPixelTag290e816a69e9439f960a9588bc2ffb54" style="display:none">
## [1] <script src="https://js.adsrvr.org/up_loader.1.1.0.js" type="text/ja ...
## [2] <script>\n            (function() {\n                var main = null ...
## {html_node}
## <noscript>
## [1] <img height="1" width="1" style="display:none" src="https://www.face ...
## {html_node}
## <script>
## {html_node}
## <noscript>
## [1] <iframe src="https://6372968.fls.doubleclick.net/activityi;src=63729 ...
## {html_node}
## <script>
```

1.3. How many of these direct descendents have an `id` attribute?

```
# Find the nodes of the body
root <- html_nodes(page, 'body')
# Getting children nodes of the body
root.child <- html_children(root)
# getting id attributes through html_attr
DD.NODES.ID <- html_attr(root.child, 'id')
DD.NODES.ID <- na.omit(DD.NODES.ID) #removing NAs

print(paste("Nodes with id attributes =", length(DD.NODES.ID), sep=""))
```

```
## [1] "Nodes with id attributes =5"
```

1.4 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.)

```
page <- read_html("https://www.yelp.com/search?find_desc=burgers&find_loc=Boston%2C%20MA&l=p%3AMA%3ABoston%3
Ads.restaurant <- page%>%
  html_nodes(".border-color--default__373c0__2oFDT:nth-child(3) .text-color--black-regular__373c0__38bRH .li
html_text()
Ads.restaurant
```

```
## [1] "Veggie Grill"
```

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.

```
library("tibble")
library("stringr")
library("rvest")

# The goal of this proble is to write a code for web scraping to get input parametrs and gives a data frame
get_yelp_sr_one_page <- function(keyword, loc="Boston, MA",page) {
# This function has 3 parameters as the input as follows:
#keyword for the search option
#location that here is set as Boston, MA (Of course it can vary)
#page number that we are looling for the result
#output is a data frame
# first step is to get the URL, it has 3 slots as "%s" which is going to be filled out with keyword, locati
YelpUrl <- 'https://www.yelp.com/search?find_desc=%s&find_loc=%s&start=%s'
page=(as.numeric(4)-1)*30
page <- as.character(page)
keyword <- "Burger"
loc <- "Boston MA"
#sprintf replaces "%s" with the 3 input parameters
YelpUrl <- sprintf(YelpUrl, URLencode(keyword), URLencode(loc), URLencode(page))
Yelp.search <- read_html(YelpUrl)

#I am using the same function throughout this HW
# This is the restaurants' name
Res.Names<-Yelp.search %>%
  html_nodes("h3 + p > a") %>%
html_text()
Res.Names <- Res.Names[-1]

# Extracting the phone numbers
Res.Phones <- Yelp.search %>%
  html_nodes(".text-align--right__373c0__3fmmn") %>%# It has the phone information
  html_text() %>%
  str_extract('[(][0-9]{3}[)] [0-9]{3}-[0-9]{4}')
Res.Phones <- Res.Phones[-1] #We remove the advertisement restaurant
Res.Phones

# Extracting the addresses
Res.Full.Address <- Yelp.search %>%
  html_nodes(".text-align--right__373c0__3fmmn") %>%
  html_text() %>%
  str_replace("[(][0-9]{3}[)] [0-9]{3}-[0-9]{4}","")
Res.Full.Address <- Res.Full.Address[-1]
# City and state, here we assume that they are Boston and MA, but in general they can be anything
Res.City <- "Boston"
Res.State <- "MA"
```

```
# Extracting the category and price levels
Res.price.categ<-Yelp.search %>%
  html_nodes(".priceCategory__373c0__3zW0R") %>%
  html_text()

Res.price.categ <- Res.price.categ[-1]

Res.Category <- Res.price.categ %>%
  str_replace('[$]+', '')
Res.Price <- str_extract(Res.price.categ, '[$]+')
Res.Price

#Scraping the number of reviews
Res.Re.Co<-Yelp.search %>%
  html_nodes(".reviewCount__373c0__2r4xT") %>%
  html_text()  %>%
  str_replace( "review[s]*","")
Res.Re.Co <- Res.Re.Co[-1]

#combining all the extracted data as a tibble data frame

Restaurant.Burger.Boston.Yelp <- tibble(Name=Res.Names, PriceLevel=Res.Price, TypeService=Res.Category, Phor
return(Restaurant.Burger.Boston.Yelp) }
Restaurants.info <- get_yelp_sr_one_page("burgers",page=3)

head(Restaurants.info)
```

```
## # A tibble: 6 x 8
##   Name    PriceLevel TypeService    PhoneNo Address  City  State ReviewsNo.
##   <chr>   <chr>      <chr>          <chr>   <chr>    <chr> <chr> <chr>
## 1 The Pa~ $$         Breakfast & B~ (617) ~ 44 Char~ Bost~ MA    "1566 "
## 2 Mystic~ $$         American (New~ (781) ~ 139 Ple~ Bost~ MA    "380 "
## 3 Charli~ $          Burgers, Beer~ (617) ~ 10 Elio~ Bost~ MA    "879 "
## 4 Eagle'~ $          Delis, Burger~ (617) ~ 1918 Be~ Bost~ MA    "410 "
## 5 The Au~ $$         Cocktail Bars~ (617) ~ 50 Hamp~ Bost~ MA    "118 "
## 6 Little~ $$         Tapas/Small P~ (617) ~ 505 Mas~ Bost~ MA    "745 "
```

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.

```
# Here, a for loop is used to scrape the data of as many pages as we want as the input to the function
get_yelp_sr_one_page <- function(keyword, loc="Boston, MA",No.Pages) {
# This function has 3 parameters as the input as follows:
#keyword for the search option
#location that here is set as Boston, MA (Of course it can vary)
#page number that we are looling for the result
#output is a data frame
# first step is to get the URL, it has 3 slots as "%s" which is going to be filled out with keyword, locati
YelpUrl <- 'https://www.yelp.com/search?find_desc=%s&find_loc=%s&start=%s'
 Multiple.pages <- tibble(Name=character(), Address=character(),PhoneNo=character(),PriceLevel=character(),

for (i in 1:No.Pages) {
YelpUrl <- 'https://www.yelp.com/search?find_desc=%s&find_loc=%s&start=%s'
```

```
page=(as.numeric(i)-1)*30
page <- as.character(page)
loc <- "Boston MA"
#sprintf replaces "%s" with the 3 input parameters
YelpUrl <- sprintf(YelpUrl, URLencode(keyword), URLencode(loc), URLencode(page))
Yelp.search <- read_html(YelpUrl)
Res.Names<-Yelp.search %>%
  html_nodes("h3 + p > a") %>%
html_text()
Res.Names <- Res.Names[-1]

Res.Phones <- Yelp.search %>%
  html_nodes(".text-align--right__373c0__3fmmn") %>%# It has the phone information
  html_text() %>%
  str_extract('[(][0-9]{3}[)] [0-9]{3}-[0-9]{4}')
Res.Phones <- Res.Phones[-1] #We remove the advertisement restaurant
Res.Phones

Res.Full.Address <- Yelp.search %>%
  html_nodes(".text-align--right__373c0__3fmmn") %>%
  html_text() %>%
  str_replace("[(][0-9]{3}[)] [0-9]{3}-[0-9]{4}","")
Res.Full.Address <- Res.Full.Address[-1]

Res.City <- "Boston"
Res.State <- "MA"

Res.price.categ<-Yelp.search %>%
  html_nodes(".priceCategory__373c0__3zW0R") %>%
  html_text()

Res.price.categ <- Res.price.categ[-1]

Res.Category <- Res.price.categ %>%
  str_replace('[$]+', '')
Res.Price <- str_extract(Res.price.categ, '[$]+')
Res.Price

Res.Re.Co<-Yelp.search %>%
  html_nodes(".reviewCount__373c0__2r4xT") %>%
  html_text()  %>%
  str_replace( "review[s]*","")
Res.Re.Co <- Res.Re.Co[-1]


#combining all the extracted data as a tibble data frame

Restaurant.Burger.Boston.Yelp <- tibble(Name=Res.Names, PriceLevel=Res.Price, TypeService=Res.Category, Phor
Multiple.pages <- rbind(Multiple.pages,Restaurant.Burger.Boston.Yelp)
}
return(Multiple.pages) }
Restaurants.info.multiple.pages <- get_yelp_sr_one_page("burgers",No.Pages=5)
head(Restaurants.info.multiple.pages)
```

```
## # A tibble: 6 x 8
##   Name    PriceLevel TypeService    PhoneNo Address  City  State ReviewsNo.
##   <chr>   <chr>      <chr>          <chr>   <chr>    <chr> <chr> <chr>
```

```
## 1 Boston~ $$         Burgers, Amer~ (857) ~ 1100 Bo~ Bost~ MA    "970 "
## 2 Tasty ~ $          Burgers, Hot ~ (617) ~ 1301 Bo~ Bost~ MA    "1113 "
## 3 Wheelh~ $$         Breakfast & B~ (617) ~ 63 Broa~ Bost~ MA    "337 "
## 4 MOOYAH~ $          Burgers, Amer~ (857) ~ 140 Tre~ Bost~ MA    "95 "
## 5 Jm Cur~ $$         American (New~ (617) ~ 21 Temp~ Bost~ MA    "787 "
## 6 Coda    $$         American (New~ (617) ~ 329 Col~ Bost~ MA    "589 "
```

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()`).

```
#Sys.sleep() has been employed inside the for loop to have a delay in the loop
get_yelp_sr_one_page <- function(keyword, loc="Boston, MA",No.Pages) {
# This function has 3 parameters as the input as follows:
#keyword for the search option
#location that here is set as Boston, MA (Of course it can vary)
#page number that we are looling for the result
#output is a data frame
# first step is to get the URL, it has 3 slots as "%s" which is going to be filled out with keyword, locati
YelpUrl <- 'https://www.yelp.com/search?find_desc=%s&find_loc=%s&start=%s'
 Multiple.pages <- tibble(Name=character(), Address=character(),PhoneNo=character(),PriceLevel=character(),

for (i in 1:No.Pages) {
YelpUrl <- 'https://www.yelp.com/search?find_desc=%s&find_loc=%s&start=%s'
page=(as.numeric(i)-1)*30
page <- as.character(page)
loc <- "Boston MA"
#sprintf replaces "%s" with the 3 input parameters
YelpUrl <- sprintf(YelpUrl, URLencode(keyword), URLencode(loc), URLencode(page))
Yelp.search <- read_html(YelpUrl)
Res.Names<-Yelp.search %>%
  html_nodes("h3 + p > a") %>%
html_text()
Res.Names <- Res.Names[-1]

Res.Phones <- Yelp.search %>%
  html_nodes(".text-align--right__373c0__3fmmn") %>%# It has the phone information
  html_text() %>%
  str_extract('[(][0-9]{3}[)] [0-9]{3}-[0-9]{4}')
Res.Phones <- Res.Phones[-1] #We remove the advertisement restaurant
Res.Phones

Res.Full.Address <- Yelp.search %>%
  html_nodes(".text-align--right__373c0__3fmmn") %>%
  html_text() %>%
  str_replace("[(][0-9]{3}[)] [0-9]{3}-[0-9]{4}","")
Res.Full.Address <- Res.Full.Address[-1]

Res.City <- "Boston"
Res.State <- "MA"

Res.price.categ<-Yelp.search %>%
  html_nodes(".priceCategory__373c0__3zW0R") %>%
  html_text()

Res.price.categ <- Res.price.categ[-1]

Res.Category <- Res.price.categ %>%
```

```
  str_replace('[$]+', '')
Res.Price <- str_extract(Res.price.categ, '[$]+')
Res.Price

Res.Re.Co<-Yelp.search %>%
  html_nodes(".reviewCount__373c0__2r4xT") %>%
  html_text()  %>%
  str_replace( "review[s]*","")
Res.Re.Co <- Res.Re.Co[-1]


#combining all the extracted data as a tibble data frame

Restaurant.Burger.Boston.Yelp <- tibble(Name=Res.Names, PriceLevel=Res.Price, TypeService=Res.Category, Phor
#Attaching every new loop to previous results by rbind
Multiple.pages <- rbind(Multiple.pages,Restaurant.Burger.Boston.Yelp)

#let the loop sleep for 3 seconds not to get banned by Yelp for abusing their website
  Sys.sleep(3)

}
return(Multiple.pages) }
Restaurants.info.multiple.pages <- get_yelp_sr_one_page("burgers",No.Pages=4)
```

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.