STAT 19000 Project 7

Topics: xml, rvest, scraping data

Motivation: There are a ton of websites that are loaded with data. However, many times the data online that we want to analyze is not readily available to download in a convenient format. Knowing how to extract data from websites in a structure way is a valuable skill, and it will enable you to explore new sources of information.

Context: In this project, we will continue to utilize principles we learned about XML files with a focus on scraping and parsing web pages. To do so, we will explore and familiarize ourselves with `rvest`. `rvest` is an R package inspired by a Python library called `beautifulsoup` for web scraping.

Scope: Understand how to scrape and parse web pages using rvest package.

You can find useful examples that walk you through relevant material here or on scholar: `/class/datamine/data/spring2020/s`
It is highly recommended to read through these to help solve problems.

Use the template found here or on scholar: `/class/datamine/data/spring2020/stat19000project07template.ipynb` to submit your solutions.

After each problem, we've provided you with a list of keywords. These keywords could be package names, functions, or important terms that will point you in the right direction when trying to solve the problem, and give you accurate terminology that you can further look into online. You are not required to utilize all the given keywords. You will receive full points as long as your code gives the correct result.

Don't forget the very useful documentation shortcut `?`. To use, simply type `?` in the console, followed by the name of the function you are interested in.

You can also look for package documentation by using `help(package=PACKAGENAME)`.

Sometimes it can be helpful to see the source code of a defined function. To do so, type the function's name without the `()`.

Question 1:

Prior to beginning, please take the time to install and load `rvest`. To do so, simply add the following to the top of your notebook:

```
install.packages("rvest")
library(rvest)
```

1a. *(1 pt)* There are two popular text editors: emacs and vim. Your friend wants your help with choosing one of these to use for an upcoming coding project. Dr. Ward told your friend to use emacs, but you are feeling brave and decided to scrape a little data to see whether other people agree with him. Dr. Ward was nice and provided you the code below to scrape reviews for 'vim' from TrustRadius, a popular website where programmers voice their opinions about programming tools. Use his code to scrape the same information for emacs. You can find emacs reviews using the following link 'https://www.trustradius.com/products/gnu-emacs/reviews'. Make sure to print out the first two results just like we do below.

```
library(rvest)
```

```
## Loading required package: xml2
```

```
vim_html <- read_html('https://www.trustradius.com/products/vim/reviews')
reviews_vim <- html_nodes(vim_html, xpath="//div[@class='review-questions']")
reviews_texts_vim <- html_text(html_nodes(reviews_vim, xpath=".//div[@class='not-edited question-text'],
```

```r
# print(reviews_texts_vim[1:2])

scores_vim <- html_nodes(vim_html, xpath="//div[@class='trust-score__score']")
scores_texts_vim <- html_text(html_nodes(scores_vim, xpath=".//span[not(@class)]"))
# print(scores_texts_vim[1:2])

likely_to_recommend_vim <- html_nodes(vim_html, xpath="//div[@class='review-questions']")
likely_to_recommend_vim_texts <- html_text(html_nodes(likely_to_recommend_vim, xpath="//div[@class='ugc
# print(likely_to_recommend_vim_texts[1:2])
```

Make sure to run the similar code for emacs as well (we want to scrape the data for both editors not just vim).

Keywords: *html_node, html_nodes, html_text, not*

Item(s) to submit:

- The code used for the problem.
- The output from printing the first 2 values of the resulting `reviews_texts_emacs` variable.
- The output from printing the first 2 values of the resulting `scores_texts_emacs` variable.
- The output from printing the first 2 values of the resulting `likely_to_recommend_emacs_texts` variable.

1b. *(2 pt)* For each of the following variables: `reviews_texts_vim`, `scores_texts_vim`, `likely_to_recommend_vim_texts`, `reviews_texts_emacs`, `scores_texts_emacs`, `likely_to_recommend_emacs_texts`, extract one relevant piece of information. For example, for the `reviews_texts_vim`, `reviews_texts_emacs`, `likely_to_recommend_vim_texts`, and `likely_to_recommend_emacs_texts`, you could create a new variable that contains the number of characters in each review, or the word count of each review, or if the reviews contain one or more of a certain word or set of words, etc. Make sure that you extract the same piece of information for each matching variable (i.e. reviews_texts_vim/reviews_texts_emacs, scores_texts_vim/scores_texts_emacs, etc.). Keep in mind that at the end the goal is to help your friend choose a text editor.

Important note: You should extract the first score from `scores_texts_emacs` and `scores_texts_vim` using `strsplit` and convert to a number using `as.numeric`.

The following is an example of extracting the word count from the `reviews_texts_vim`, and `reviews_texts_emacs` variables:

```r
# This is an example of extracting a relevant piece of information.
# Please note that the solution should have a total of 6 "new" sets of data.

# This is a possible solution for `reviews_vim`
count_words <- function(x) {length(strsplit(x, " ")[[1]])}
reviews_vim_word_count <- unname(sapply(reviews_texts_vim, count_words))
# print(reviews_vim_word_count)

# This is a possible matching solution for `reviews_emacs`
reviews_emacs_word_count <- unname(sapply(reviews_texts_emacs, count_words))
# print(reviews_emacs_word_count)

# A solution for `scores_texts_vim` should go under this

# A solution for `scores_texts_emacs` should go under this

# A solution for `likely_to_recommend_vim_texts` should go under this
# A solution for `likely_to_recommend_emacs_texts` should go under this
```

Hint: *Take a look at the examples, and think about how you would use each variable to make a recommendation.*

Note: *Keywords will vary depending on what you decided to extract from the variables.*

Potential keywords: *as.numeric, substr, grep*

Item(s) to submit:

- The code used for the problem.
- The output from printing the "new" sets of data for each of: `reviews_texts_vim`, `scores_texts_vim`, `likely_to_recommend_vim_texts`, `reviews_texts_emacs`, `scores_texts_emacs`, and `likely_to_recommend_emacs_texts`.

1c. *(2 pt)* In this question, we will combine all of the data from (1a) and (1b) into a single `data.frame`. This makes it easy for us to thoroughly explore the data, plot the data, and use it in functions. The end result will resemble the following:

```
df <- data.frame(editor=c('vim', 'vim', 'emacs', 'vim', 'emacs'), reviews=c("this is the first review fo
print(df)
```

```
##   editor                             reviews scores_texts word_count
## 1    vim    this is the first review for vim       4 of 5          7
## 2    vim   this is the second review for vim       5 of 5          7
## 3  emacs  this is the first review for emacs       3 of 5          7
## 4    vim    this is the third review for vim       4 of 5          7
## 5  emacs this is the second review for emacs       5 of 5          7
```

As you can see, the `editor` column dictates which editor the row's data is for: vim or emacs. Please note that the end result will have more columns, this is just an example. To combine all of your data, first create two data.frames. One called `df_vim` and another called `df_emacs`. Each data.frame should have the `editor` column filled with the name of the editor. For example, `df_vim` should have the value "vim" for each and every row of the data.frame. Add the remaining variables from (1a) and (1b) to each respective dataframe as new columns, and make sure the column names match exactly.

```
df_vim <- data.frame(editor="vim", review_word_count=c(1,2,3,4,5))
print(df_vim)
```

```
##   editor review_word_count
## 1    vim                 1
## 2    vim                 2
## 3    vim                 3
## 4    vim                 4
## 5    vim                 5
```

```
df_emacs <- data.frame(editor="emacs", review_word_count=c(5,4,4,4,3,2,1))
print(df_emacs)
```

```
##   editor review_word_count
## 1  emacs                 5
## 2  emacs                 4
## 3  emacs                 4
## 4  emacs                 4
## 5  emacs                 3
## 6  emacs                 2
## 7  emacs                 1
```

Notice how each data.frame has the exact same column names? Continue this pattern and add the rest of your variables to each data.frame. Once completed, run the following to combine the data.frame's:

```r
# Note that rbind relies on the column names in order to combine the data.
final_df <- rbind(df_vim, df_emacs)
print(final_df)
```

```
##    editor review_word_count
## 1     vim                 1
## 2     vim                 2
## 3     vim                 3
## 4     vim                 4
## 5     vim                 5
## 6   emacs                 5
## 7   emacs                 4
## 8   emacs                 4
## 9   emacs                 4
## 10  emacs                 3
## 11  emacs                 2
## 12  emacs                 1
```

```r
# What that means is that as long as you have all of the columns named exactly the same, rbind
# will properly combine your data!
vim_reversed <- df_vim[, c(2,1)]
still_good_df <- rbind(vim_reversed, df_emacs)
print(still_good_df)
```

```
##    review_word_count editor
## 1                  1    vim
## 2                  2    vim
## 3                  3    vim
## 4                  4    vim
## 5                  5    vim
## 6                  5  emacs
## 7                  4  emacs
## 8                  4  emacs
## 9                  4  emacs
## 10                 3  emacs
## 11                 2  emacs
## 12                 1  emacs
```

Great now you have your data.frame, `final_df` to use in question 2!

Keywords: *data.frame*

Item(s) to submit:

- The code used for the problem.
- The output from printing the `head` of the `final_df`. This output should have at least 7 columns: `editor`, `reviews_texts`, `scores_texts`, `likely_to_recommend_texts`, as well as the "new" sets of data from (1b), which, when combined should result in 3 additional columns.

Question 2:

We have now scraped, parsed, and organized data on vim and emacs into a single data.frame called `final_df`. Although it is not a lot of data, you can imagine a situation where a website contains thousands of entries.

Let's do some data exploration and see if we can find any interesting relationships in our data.

Hypothesis: *(1 pt)* Propose a hypothesis for how you think two variables are related (if at all). Make the numeric score you extracted in (1b) one of the two variables you will draw a hypothesis about.

Let's say, for example, that you have the variable `word_count` that has the number of words in each review, and another variable called `character_count` that has the number of characters (letters/numbers/symbols) in each review. A good hypothesis would be: "I hypothesize that `word_count` and `character_count` are positively correlated." (meaning the more words that a review has, the more characters the review will have). This one is obvious, as typically the more words that a review has the more characters it will have. Note that we could have hypothesized: "I hypothesize that `word_count` and `character_count` are negatively correlated." In our exploratory analysis step, however, we would discover that is most likely not true. A wrong hypothesis is still valid.

Exploratory analysis: *(3 pts)* Poke through the information like we did in the examples. Include short comments about what you are doing and why for each step. Make sure you explore your data in at least two different ways.

The following are some examples of ways to explore your data:

- getting a correlation coefficient using `cor`
- plotting a scatterplot using `plot`
- plotting a lineplot using `plot(..., type='l')`
- making a barplot using `barplot`
- evaluating the median values of columns in our data using `sapply`
- breaking our data into groups and calculating statistics on those groups using `tapply`
- etc.

You are by no means limited to just these methods, you may do anything you'd like, just make sure to mention *what* and *why* in your comments.

Final recommendation: *(1 pt)* Write down your final recommendation (vim or emacs), 1-2 sentences about whether or not you noticed anything interesting in your exploratory analysis, and any other comment you may have.

Don't worry, your conclusion will not effect your grade. We have a very small sample here, and the important thing is to learn something new and understand how scraping data is a useful tool to have in your toolbox! Have fun and be creative!

Hint: *Take a look at the examples for some ideas.*

Item(s) to submit:

- A markdown cell with your hypothesis.
- The code used for the exploratory analysis where you explored the relationship between the numeric score from (1b) and another variable. Each line of code should have a comment explaining what is going on.
- A markdown cell with your final recommendation (vim or emacs), 1-2 sentences about whether or not you noticed anything interesting in your exploratory analysis, and any other comment you may have.

Project Submission:

Submit your solutions for the project at this URL: https://classroom.github.com/a/PCJeRdng using the instructions found in the GitHub Classroom instructions folder on Blackboard.

Important note: Make sure you submit your solutions in both .ipynb and .pdf formats. We've updated our instructions to include multiple ways to convert your .ipynb file to a .pdf on scholar. You can find a copy of the instructions on scholar as well: `/class/datamine/data/spring2020/jupyter.pdf`. If for some reason the script does not work, just submit the .ipynb. Make sure you have your output calculated and displayed inside your notebook prior to submission.