STAT 19000 Projects

Topics: R, SQL, and associated tools

Motivation: Practice, practice, practice. Continuing to learn about and use the tools we've addressed this semester will make you faster and more efficient. You will be able to break problems into logical parts, and will discover you get "stuck" less often.

Context: We've explored a wealth of tools like R, SQL, and bash. We are now going to use what you've learned to solve a variety of different problems.

Scope: The scope of this project encompasses all topics covered this semester, namely, R, SQL, and associated tools.

Important note:

> Please remember that we assigned 12 projects this semester, and students are able to *drop* their lowest two scores, so that only 10 projects are included in the overall grading. This is the same scheme as we had during the fall semester.
> Also (same as during the fall semester), we will not have a final exam. Instead, we've provided 4 "optional" projects (below), which can be use as substitutes for any of the 10 required projects. You do not need to do any of these optional make-up projects, if you are happy with your grades from 10 out of the regular 12 projects, but you are welcome to do as many of them as you like. (A make-up project is simply used for a grade replacement.)

——— Optional Project 1 ———

#R #scraping

Question 1: goodreads

First, load the package `rvest` using the command below: `library(rvest)`

1a. *(1 pt)* Navigate to https://goodreads.com and find your favorite book. What is the numeric, goodreads "id" for your favorite book?

Test to make sure that you really do only need the numeric portion of the id.

Example: The first book in my favorite series is "The Eye of the World". The goodreads "id" appears to be 228665. If I navigate to https://www.goodreads.com/book/show/228665 instead of https://www.goodreads.com/book/show/228665.The_Eye_of_the_World they go to the same page, which indicates we don't need the ".The_Eye_of_the_World" part of the url.

> **Item(s) to submit:**
> - The goodreads "id" for your favorite book
> - The url for your favorite book

1b. *(1 pt)* Let's scrape some data for your favorite book!

We provided you with two functions as examples: `get_number_reviews` and `get_book_name`. Here is how they work:

```
get_number_reviews <- function(book_html){
        number_reviews_node <- html_nodes(book_html, xpath="//meta[@itemprop='reviewCount']")
        number_reviews <- as.numeric(html_attr(number_reviews_node, 'content'))
        return(number_reviews)
}

get_book_name <- function(book_html){
        title_node <- html_nodes(book_html, xpath="//h1[@id='bookTitle']")
        title_text <- gsub("[\r\n]", "", html_text(title_node))
        return(trimws(title_text))
}

# our get_number_reviews & get_book_name functions accept objects like this:
our_html_page <- read_html("https://www.goodreads.com/book/show/228665")

# read_html is scraping the data from the webpage itself. Instead of passing
# our functions a goodreads id and scraping the webpage on each function call,
# we scrape the data once, and pass the data to our functions. Our functions
# simply parse the web page!
get_book_name(our_html_page)
```

```
## [1] "The Eye of the World"
```

```
get_number_reviews(our_html_page)
```

```
## [1] 11516
```

Finish the functions `get_avg_rating` and `get_number_ratings` provided below by replacing "YOUR LINE OF CODE HERE" with the actual line of code.

We want `get_avg_rating` to get the average rating of the book you provide. We want `get_number_ratings` to get the number of ratings that the book has received.

```
get_avg_rating <- function(book_html){
        ratingValue_node <- html_nodes(book_html, xpath="//span[@itemprop='ratingValue']")
        ratingValue_text <- YOUR LINE OF CODE HERE
        ratingValue_number <- YOUR LINE OF CODE HERE
        return(ratingValue_number)
}

get_number_ratings <- function(book_html){
        number_ratings_node <- html_nodes(book_html, xpath="//meta[@itemprop='ratingCount']")
        number_ratings_att_text <- YOUR LINE OF CODE HERE
        number_ratings_as_number <-  YOUR LINE OF CODE HERE
        return(number_ratings_as_number)
}
```

Use the finished versions of `get_avg_rating` and `get_number_ratings`, as well as the provided functions `get_number_reviews` and `get_book_name` to scrape information from the url you chose in (1a). Make sure you first read your url using the function `read_html` before passing it to the functions.

Keywords: *html_text*, *as.numeric*, *html_attr*.

Hint: *Start by printing the first lines in the functions, this will help you understand what you are working*

with:

```
book_html <- read_html("https://www.goodreads.com/book/show/228665")
ratingValue_node <- html_nodes(book_html, xpath="//span[@itemprop='ratingValue']")
number_ratings_node <- html_nodes(book_html, xpath="//meta[@itemprop='ratingCount']")
print(ratingValue_node)
print(number_ratings_node)
```

Hint: *Note that in* `get_avg_rating` *we want to get the text from the html node, and then transform it to a numeric format. In* `get_number_ratings` *we want to get the value from the attribue named 'content', and then transform it to a numeric format.*

Hint: *Run* `as.numeric('\n 2 \n')` *and* `as.numeric('2')`*. They should give the same result.*

Example: *You can test your functions on https://www.goodreads.com/book/show/228665:*

```
book_url <- "https://www.goodreads.com/book/show/228665"
get_avg_rating(read_html(book_url)) # (should be 4.19)
get_number_ratings(read_html(book_url)) # (should be 361,232)
get_number_reviews(read_html(book_url)) # (should be 11,429)
get_book_name(read_html(book_url)) # (should be "The Eye of the World")
```

Item(s) to submit:

- A code chunk with the finished `get_avg_rating` and `get_number_ratings` functions.
- The results from `get_avg_rating`, `get_number_ratings`, `get_book_name`, and `get_number_reviews` applied to your favorite book's goodreads url.

1c. *(1 pt)* Navigate to "The Best Epic Fantasy (fiction)" webpage. As you can see this webpage contains a list of 100 books, with "next" links to get even more books (34 pages worth!). Run the code below to get a list of goodreads id's for each book in the list. The results are saved in the variable `epic_books_pages`.

```
# the first page, containing 100 books
epic_books_URL <- 'https://www.goodreads.com/list/show/50.The_Best_Epic_Fantasy'

# reading the list
epic_books_list <- read_html(epic_books_URL)

# get the maximum number of pages (that contain links to more books)
epic_books_list_pages <- html_nodes(epic_books_list, '.pagination a')
last_page_number <- max(as.numeric((html_text(epic_books_list_pages))), na.rm=T)

# get the urls for each and every page
epic_books_pages <- sapply(1:last_page_number, function(page_number){
        paste(epic_books_URL,"?page=",page_number,sep='')
})
```

Below we have a function called `get_book_ids`. It takes a url (from our `epic_books_pages` list) and returns the goodreads book ids for every book in the list on the provided page. Use the `epic_books_pages` variable and the provided `get_book_ids` function to get the book id for every book in the "The Best Epic Fantasy (fiction)" list. Name this variable `book_ids`. This may take a while (a few minutes), grab a coffee while it runs.

```
get_book_ids <- function(list_page_url){
        list_page_html <- read_html(list_page_url)
        node_containing_id <- html_nodes(list_page_html, xpath="//td//div[@class='u-anchorTarget']")
        book_id <- html_attr(node_containing_id, 'id')
        return(as.numeric(book_id))
```

```
}
```

`book_ids` will be a list if you set `simplify` to `F` inside your `sapply`. Use the following code to "flatten" the list:

```
book_ids <- unname(unlist(book_ids))
```

keywords: *sapply*

> **Item(s) to submit:**
> - The code you used to get `book_ids`.

1d. *(2 pts)* Now that we have the all the book's "id"s in `book_ids`, we can use the functions from (1b) to scrape rating information. We combined the functions in (1b) in a single function called `get_book_info`. Our function uses the book id to create a url, reads it in, and scrapes the rating information. Run the code below to get a data.frame containing the book id, its position on the list, and the information scrapped using the functions in (1b).

To scrape several pages, we need to add a pause so goodreads doesn't get mad at us. For simplicity, we will only get rating information on the first 10 and last 10 books on the list. This code will take a while, so if you haven't yet, make some coffee while it runs.

```
# function combining information from (1b)
get_book_info <- function(book_id){
       # to ensure we can scrape all the information:
       Sys.sleep(10) # pause for connection to work
       closeAllConnections()
       book_url <- paste('https://www.goodreads.com/book/show/',book_id, sep='')
       book_html <- read_html(book_url)
       name <- get_book_name(book_html)
       avg_rating <- get_avg_rating(book_html)
       n_ratings <- get_number_ratings(book_html)
       n_reviews <- get_number_reviews(book_html)
       return(data.frame(book_id, name, avg_rating, n_ratings, n_reviews))
}

# getting information for first 10 and last 10 books
epics_books_df <- sapply(book_ids[c(1:10, 3354:3364)], get_book_info, simplify=FALSE)

# Combining it into a data.frame
epics_books_df <- do.call(rbind, epics_books_df)
epics_books_df <- data.frame(epics_books_df,
                             pos = c(1:10, 3354:3364))
```

Now we have the data organized in a data.frame, `epic_books_df`, and can do some analysis. Check whether (in your opinion) the number of ratings and number of reviews are associated (have a relationship) by creating a scatterplot.

> **Item(s) to submit:**
> - One line of code with your solution to get scatterplot.
> - 1-2 sentences explaining whether you think there is an association or not between the number of ratings and the number of reviews, and why.

1e. *(optional, 0 pts)* Present an interesting factoid or graphic based on `epic_books_df`. Be creative and have fun!

Item(s) to submit:
- Code chunk used to obtain intereseting factoid or graph.
- Resulting factoid or graph.
- 1-2 sentences discussing why do you think this is an interesting factoid or graph.

Question 2: create a question

2a. *(3 pts)* The entire internet is yours for the taking. Find a website with data you are interested in scraping. You have two options:

1. Write a question that asks the user to write a function to scrape info from the website given an id of some sort. Based on the id, the function will fetch a particular set of data.
2. Write a question that asks the user to scrape a set of data. Unlike option (1), this question should require the scraped data to be of a significant size – not merely a few factoids.

For each option, provide the answer to your question, including any code you used. In addition to solving the question using R code, be sure to explain your method or methods used in the solution.

Item(s) to submit:
- Your newly created question in markdown within your RMarkdown file.
- A code chunk in the same RMarkdown file with the solution to your question.

2b. *(2 pts)* Create an interesting plot or graphic using the data you scraped in (2a). If option (1) was chosen, use your function to scrape info for at least 5 "ids", and use the resulting data to create your graphic.

Item(s) to submit:
- A `jpeg` or `png` image with your newly created graphic.
- As you are not limited on *how* you create your graphic, you are not required to submit any code for this question.

——— Optional Project 2 ———

#R #sql

Rotten tomatoes vs. IMDB

IMDB and rotten tomatoes are common websites used to check ratings for movies and TV shows. Some people prefer one to the other, and have their reasons for it. Which one do you prefer? In project 11 we analyzed an IMDB database. In Project 12 we created an SQL database for rotten tomatoes. Let's use some data to compare how the ratings between the different websites compare.

First, load the package `RSQLite` using the commands below:

```
library(RSQLite)
```

Recall that the databases are located on scholar:

`/class/datamine/data/spring2020/imdb.db`

and

`/class/datamine/data/spring2020/rt.db`

1a. *(1 pt)* Create two connections, one with the imdb database, and one with rotten tomatoes database. Called them, respectively, `imdb_connection` and `rt_connection`. Test that your connection works by running the following commands:

```
# (Should return a vector containing: akas, crew, episodes, people and ratings)
dbListTables(imdb_connection)

# (Should return a vector containing: movies, reviews)
dbListTables(rt_connection)
```

Keywords: *dbConnect, RSQLite::SQLite()*

Item(s) to submit:
- The code chunk used to create a connection with both the imdb and rotten tomatoes databases.

1b. *(1 pt)* To compare the ratings, we must first organize the IMDB database. Join the movie titles from the `titles` table with their corresponding rating information from the `ratings` table by the `title_id` column in both tables. Filter only on titles of type "movie". Call the resulting dataframe `imdb_movie_ratings`.

Keywords: *inner join*, *on*, *and*

Hint: *Your dataset should have 245,016 rows.*

Hint: As we want to compare IMDB and Rotten Tomatoes ratings, we only want information on movie's that exist in both tables.

Item(s) to submit:
- Code chunk used to create `imdb_movie_ratings`.
- The first 5 movies in `imdb_movie_ratings`.

1c. *(2 pts)* Let's start with a simple comparison between the two websites. Get the maximum, minimum and average rating for both IMDB and rotten tomatoes movies with genre LIKE `Drama` – if the genre has the word "Drama" anywhere in it, we'll consider it like `Drama`. Specifically, we want to look at the `audience_rating` in the `movies` table in the rotten tomatoes database. We want to look at the `rating` column from the imdb database's `ratings` table. In the IMDB database table `ratings`, the genre column is named `genres`. In the Rotten Tomatoes `movies` table, he genre column is named `genre`.

Is there a difference between the two websites based on this summary? Note that IMDB ratings are between 0 and 10, while Rotten Tomatoes are between 0 and 100. This should be taken into account when making the comparisons. You *do not* need to multiply the IMDB ratings be 10, just draw a conclusion based on what you see.

Keywords: *max*, *min* **Keywords (SQL):** *avg*, *where*, *like* **Keywords (R if using `imdb_movie_ratings`):** *subset*, *mean*, *grepl*

Hint: To select only rows in `mytable` for which `column10` contains `specific_word`, use the SQL command `SELECT * FROM mytable WHERE column1 LIKE '%specific_word%';`

Item(s) to submit:
- Code chunk(s) used to get the maximum, minimum and average rating for imdb and rotten tomatoes database.
- Maximum, minimum and average rating values for imdb and rotten tomatoes database.
- 1-2 sentences comparing the websites based on the summary rating information.

1d. *(2 pts)* As rotten tomatoes has two ratings, `tomatometer_rating` and `audience_rating`, one could want to average them before comparing to IMDB. Get the maximum, minimum and average values for a combined

rating for Rotten Tomatoes movies with genre `Drama`. Let the combined rating be the average between the `tomatometer_rating` and the `audience_rating`. How does IMDB and Rotten Tomatoes compare based on this new summary?

Keywords: *+*,

Hint: You can sum column1 and column2 in SQL using `column1+column2`.

Item(s) to submit:
- Code chunk(s) used to get the maximum, minimum and average combined rating for rotten tomatoes database.
- Maximum, minimum and average rating values for rotten tomatoes combined rating.
- 1-2 sentences comparing the websites based on the summary rating information.

1e. *(2 pts)* So far we've compared IMDB and rotten tomatoes ratings on summary information (mean, max, min, etc). How do the ratings compare per movie for `Drama` movies with runtime greater or equal to 90 minutes?

Create two datasets: `rt_drama_movies_dataset`, and `imdb_drama_movies_dataset` containing the rating information for `Drama` movies with at least 90 minutes of runtime. To create `rt_drama_movies_dataset` use the table `movies` from the rotten tomatoes database. To create `imdb_drama_movies_dataset` you can use the tables `ratings` and `title` from the IMDB database, or use the `imdb_movie_ratings` dataframe from (1b).

Item(s) to submit: - Code chunk(s) used to get `imdb_drama_movies_dataset` and `rt_drama_movies_dataset`.

1f. *(2 pts)* Run the code chunk below to merge the datasets `imdb_drama_movies_dataset` and `rt_drama_movies_dataset`.

```
drama_movies <- merge(imdb_drama_movies, rt_drama_movies,
                      by.x='primary_title', by.y='movie_title')
```

Note that the `rating` for IMDB is now called `rating.x` as `rt_drama_movies` also contained a column named `rating`.

Create a scatterplot comparing the IMDB ratings, `rating.x`, multiplied by 10 and the rotten tomatoes audience rating, `audience_rating`. How do the websites compare? Which one do you prefer?

Item(s) to submit: - Code chunk(s) used to get scatterplot. - Scatterplot comparing IMDB ratings (multiplied by 10) and rotten tomatoes audience ratings for `Drama` movies. - 1-2 sentences comparing the websites based on the scatterplot. - 1-2 sentences explaining which website you prefer (if any), and why.

Optional Project 3

#R #RMarkdown

During this semester you've gained exposure to a variety of tools to assist with data analysis: web scraping, database manipulation, and proper documentation to create reproducible analysis are among the skills learned. Properly showcasing key learnings is a valuable skill. It comes in handy not only for class presentations, but also during your career. One example that many of you will face is presenting your work at the end of an internship.

Copy the RMarkdown slide template found here or on scholar:

`/class/datamine/data/spring2020/stat19000optionalproject03_template.Rmd`

and rename it to `finalproject.Rmd`. Modify the RMarkdown template to create a slide presentation demonstrating what you've learned from STAT19000 during the spring semester of 2020. Your presentation must include (but is not limited to) the following:

1. *(0.5 pts)* Your name. Be sure to replace "Your_name_here" with your name in the author information.
2. *(0.5 pts)* A proper, descriptive title for every slide. Be sure to replace "Your_slide_title" with an appropriate title for each slide.
3. *(0.5 pts)* Your first slide must be a summary of (at least) 4 summarized key subjects. An example of a key subject is how to use the apply suite.
4. *(6 pts)* Each topic must have at least 3 slides:
    1. One slide with 1-2 sentences explaining the topic. For the apply suite example, 1-2 sentences explaining what are the apply suite functions and when we use them, would suffice.
    2. At least 2 "show cases" of the topic. For example, if your topic is related to the apply suite, provide (at least) 2 chunks of R code with results exemplifying the use of apply functions.
    3. 1-2 sentences explaining why you think this topic is revelant to you.
5. *(1.5 pts)* One slide containing 3+ recommendations and/or suggestions for future STAT19000 classes.
6. *Optional* One slide containing what you enjoyed about STAT19000, and believe we should keep doing for future semesters.

Once you've modified the file, you'll find you have *new* rendering choices: ioslides (html), slidy (html), and beamer (pdf). Pick which one you like best and use that for your submission.

Item(s) to submit:

- Modified `finalproject.Rmd` file.
- Rendered `finalproject.html` file or `finalproject.pdf` file.

——— Optional Project 4 ———

#R #applysuite

Question 1: The Office lines

We've provided you with a dataset for the TV show The Office here or on scholar:

`/class/datamine/data/spring2020/the_office_dialogue`.

Use `read.csv()` to read the dataset into a dataframe. Make sure to use `stringsAsFactors=FALSE` when reading the data. Call this dataset `the_office_data`.

1a. *(2 pts)* Create a function that takes two inputs, `season_number` and `character_name`, and calculates the total number of lines of character with `character_name` for the season with `season_number`. Call this function `get_number_lines`. Test your functions on the following commands:

```
get_number_lines("Jim", 1) (should be 228)
get_number_lines("Kelly", 3) (should be 123)
get_number_lines("Angela", 2) (should be 135)
get_number_lines("Dwight", 1) (should be 208)
```

Here is a skeleton function:

```
get_number_lines <- function(character_name, season_number) {
    # write code that counts the number of lines character_number has in season_number
    # note that each line in the data corresponds to a line

    # number_of_lines should be a number
    return(number_of_lines)
}
```

Keywords: *function, subset, tapply, length, nrow*

Hint: *Use the* `character_name` *to create a subset of the dialogue dataset that contains information only for the* `character` *with* `character_name`*, and* `season` *with* `season_number`*. Calculate how many lines this charater had in this season. Note that each row corresponds to a line.*

Item(s) to submit:
- The complete `get_number_lines` function.

1b. *(1 pt)* Modify your `get_number_lines` function to take a single input `character_name`, and to calculate the number of lines for `character_name` for all seasons. Test your functions on the following command:

`get_number_lines("Jim")` The answer should be:

1	2	3	4	5	6	7	8	9
228	752	690	575	840	928	686	806	798

Here is a skeleton function:

```
get_number_lines <- function(character_name) {
    # write code that counts the number of lines character_number has for
    # each season.
    # note that each line in the data corresponds to a line

    # number_of_lines should be a number
    return(number_of_lines)
}
```

Keywords: *tapply, length*

Hint: *Make sure to modify your function to subset by only* `character_name`*.*

Item(s) to submit: - The completed, updated `get_number_lines` function.

1c. *(2 pts)* Use your (1b) `get_number_lines` function to create a matrix containing the number of lines per season for the following characters: Dwight, Pam, Jim, and Angela.

Keywords: *sapply*

Hint: *You may want to create a vector containing the name of the characters you want to evaluate.*

Item(s) to submit:
- Code used to create desired matrix.

1d. *(2 pts)* Create a vector called `season_average_rating` that contains the average imdb rating of the season. Note that the `imdb_rating` in the dataset is per episode.

Keywords: *mean, tapply*

Hint: *You may want to first create a dataset that contains only unique values for the following information:* `season`*,* `episode`*, and* `imdb_rating`*. To do so, run the command:*

`imdb_data <- unique(the_office_data[,c('season', 'episode', 'imdb_rating')])`.

Hint: *Remember to set the* `na.rm` *to* `TRUE` *when calculating the mean.*

Item(s) to submit:
- Code chunk used to create `season_average_rating`
- Results for `season_average_rating` vector

Question 2: Graphing The Office

2a. *(1 pt)* Dwight Schrute is one of the most popular characters in The Office. Let's explore whether seasons with higher Dwight dialogues have higher ratings. Make a scatterplot comparing `season_average_rating` and Dwight's total number of lines calculated in (1c). Based on the newly created graph do you think there is any association between Dwight's dialogue and The Office ratings?

Item(s) to submit:
- Scatterplot for `season_average_rating` and Dwight's total number of lines calculated in (1c).
- 1-2 sentences explaining if you think there is any association between Dwight's dialogue and The Office ratings, and why.

2b. *(2 pts)* Present an interesting factoid or graphic from The Office dataset. Be creative and have fun! You can explore the ratings of different episodes and seasons, compare ratings with dialogue between certain characters, compare number of lines per season for different characters, etc.

Item(s) to submit:
- Code chunk used to obtain intereseting factoid or graph.
- Resulting factoid or graph.
- 1-2 sentences discussing why do you think this is an interesting factoid or graph.

Project(s) Submission:

Submit solutions for all projects using the instructions found in the GitHub Classroom instructions folder on Blackboard.

You do not need to do any of these make-up projects if you are happy with your grades on your top 10 projects. A make-up project would replace whatever your lowest score is (which could be a project you didn't do).

Submit make-up projects using the following links:

Optional Project 1: https://classroom.github.com/a/mzTFb23e

Optional Project 2: https://classroom.github.com/a/_1Y-Dzy7

Optional Project 3: https://classroom.github.com/a/T3aWz30j

Optional Project 4: https://classroom.github.com/a/C9fowzmR