Lab/HW 4: Scraping - Data from the Web

Your lab/homework must be submitted in with two files: (1) R Markdown format file; (2) a pdf or html file. Other formats will not be accepted. Your responses must be supported by both textual explanations and the code you generate to produce your result.

Part 0 - Regular expressions - Miscelleneous Exercises - Extra Credit

In the following exercises, please think of a good way to demonstrate that you found the correct answer. Note that you may use `str_view()`, it's HTML output will be combined in the document that RMarkdown compiles (if you choose to work with HTML output)

Each of the following questions is worth 1 extra credit points (14 total)

1. How would you match the literal string $^$ (not as a part of another string)?

Repetition

1. Create a regular expression to find all words from `stringr::words` have three or more vowels in a row.
2. Create a regular expression to find all words from `stringr::words` have two or more vowel-consonant pairs in a row.

Grouping and backreferences

1. Create a regular expression to find all words from `stringr::words` that start and end with the same character.
2. Create a regular expression to find all words from `stringr::words` that contain a repeated pair of letters (e.g. "church" contains "ch" repeated twice.)
3. Create a regular expression to find all words from `stringr::words` that contain one letter repeated in at least three places (e.g. "eleven" contains three "e"s.)

Detect matches

1. Create a regular expression to find all words from `stringr::words` that start with a vowel and end with a consonant.
2. Are there any words from `stringr::words` that contain at least one of each different vowel?
3. What word from `stringr::words` has the highest proportion of vowels?

Extract matches

1. From the Harvard sentences data (`stringer::sentences`), extract the first word from each sentence. (word = any contiguous set of letters)
2. From the Harvard sentences data (`stringer::sentences`), extract all words ending in `ing`.

Replacing matches

1. Switch the first and last letters in `stringr::words`.
2. Which of those strings are still words (i.e. still included in `stringr::words`)?
3. Of these, which don't start and end with the same letter?

Part I - Regular expressions - Scraping HTML to data frame

[35 pt] [5 pt] 1. Use the `readLines` command to load the file into a character vector called `richhtml`. How many lines does it contain? What is the total number of characters in the file?

[5 pt] 2. Open the file in a text editor (*not* as a web-page). Find the entries for Bill Gates and for Stanley Kroenke. Give the text of the lines from the file that has their names and their net worths.

[25 pt] 3. Use regular expressions to extract the data from the web page and build a data frame that has 2 columns: name and net worth in billions of dollars. The name column should be characters and the net worth column should be numeric. There should be 100 rows.

Part II - Regular Expressions - Scraping to other structures

[50 pt] In this part, we will write our own "auto-complete" / "spelling suggestions" function. Our raw data is an XML file that contains 4,938 tweets that include the string "UMass" (the file is supplied on Moodle). We will use the combined words and their respective frequencies to suggest auto-completion choices. That is, the input for our final function will be a character object, say `"he"`, and an integer, say `10`. The output will be a character vector at the length of the input integer, with several suggestions for completing the word, the first such word will be the most frequent word in the database that starts with `"he"`, the second word will be the second most frequent word in the database that starts with `"he"`, etc. If there are not enough words in in the tweets dataset, the missing ones will be filled according to frequency from a table containing the 10000 most frequent words in the English vocabulary (from https://en.wiktionary.org/wiki/Wiktionary:Frequency_lists/PG/2006/04/1-10000)

For instance, based on our dataset and vocabulary, the output for `"sh"` and `10` will be the vector `c("should", "show", "she", "shit", "share", "shot", "shuttle", "short", "shaycormac_1", "shawmut")` (see more examples below).

The following workflow is provided as a suggestion only. If you write your code differently and perform the same tasks, you will receive full credit.

In order to receive full credit for the assignment, you need to provide an explanation in your words for every section that is formulated as a question. E.g. for 1b it is not enough to print the contents of the first line. You have to explain what you see there.

1. Assign the raw dataset to a variable named `umass_tweets_xml_raw`:

```
umass_tweets_xml_raw <- readLines("umass_tweets_raw.xml")
```

using the function `head()` and `tail()` examine its first and last lines.

a. What is the type of data of `umass_tweets_xml`? What is it's size?
b. What does the first line contain?
c. What does the second line contain?
d. What does the third line contain?
e. What does the second to last line contain?
f. What does the last line contain?
g. Create a regular expression that is unique to the lines that have messages and verify that it captures the 4,938 tweets.
h. using that regular expression and function/s that we discussed in class, assign only the rows of `umass_tweets_xml` that have messages in them to a new variable `umass_tweets_xml`.

2. Our next task is to extract just the full text of each tweet. The first one, for example, is:

UMass Amherst is hiring an assoc / full prof of race, media, and comm who can "draw from critical bodies of theory and epistemologies of Black, Latino/a/x, Asian, Indigenous, and Ethnic Studies" https://t.co/GUHy5gvtdO

We will store each tweet as a vector of separate words.

a. define a regular expression that matches everything that is **not** part of the full text of a tweet.
b. use a function to substitute `""` for everything that is not a part the full text in `umass_tweets_xml` (one line of code)
c. using another function and the helper `boundary()` create a variable which will store all the words from the text messages (for our needs, it suffices to identify words as being separated any type of space). In addition, convert all words to lowercase (hint that may assist you: the `unlist` function).

3. We will now create our "Twitter vocabulary" - a vector where each word is associated with a count.

a. Use one of the functions we discussed in class to create a variable named `twitter_vocab` that contains a table of all words and their counts. Tip: you can use the function `sort()` with the option `decreasing = TRUE`.
b. save your twitter vocabulary using:

```
saveRDS(twitter_vocab, "twitter_vocabulary.rds")
```

4. Let us now process the file "top_10000_english_words.txt" so that it will be useful for us. The file is downloaded in raw text format. Read it to R using `english_vocab_raw <- readLines("top_10000_english_words.txt")` and using regular expressions create a character vector named `engilsh_vocab` of length 10,000, where each entry is a word, and those are organized from high to low frequency.

Save your English vocabulary using:

```
saveRDS(english_vocab, "english_vocabulary.rds")
```

5. We are now ready to write our auto-complete function!

a. Write a function that takes as inputs a character value and an integer.
b. Using `twitter_vocab <- readRDS("twitter_vocabulary.rds")` the function will load the twitter vocabulary you pre-processed.
c. Using the `paste` or `str_c` function it creates a regular expression that will identify all words that start with the input characters.
d. Using one of the functions we worked with in class and the above regular expression, identify all the locations in the `names(twitter_vocab)` vector (which corresponds to the words) that start with the input characters.
e. select all the entries of `twitter_vocab` based on the above locations, make sure that they are sorted according to frequency so that the most frequent are at the head of the vector, choose the top ones (according to the integer input of this function) and return the relevant `names` which should correspond to the most frequent words.
f. if there are not enough words that start with the input character from the twitter vocabulary load the english vocabulary using `english_vocab <- readRDS("english_vocabulary.rds")` and find there the most frequent words that start with the input string (and are not already identified by the twitter vocabulary!) and add those as needed. If words are added from the English dictionary, please use the function `message()` to inform the user that this happened, and which words were added from it. If the English vocabulary too leaves you with fewer than the requested number of words, return a shorter vector (don't return NA values!).

Some results: **Demonstrate that your function provides the exact same results**

1.

```
spellingSugestion("sh", 15)
```

```
##  [1] "should"       "show"         "she"          "shit"         "share"
##  [6] "shot"         "shuttle"      "short"        "shaycormac_1" "shawmut"
## [11] "shirt"        "shots"        "shown"        "shows"        "shame"
```

15 words, all from the Twitter data set.

2.

```
spellingSugestion("baro")
```

```
## Added the following English vocabulary words: baron, baronet
```

```
## [1] "barometer"      "barometers"     "baron_scicluna" "baron"
## [5] "baronet"
```

Five words (where 10 were requested), two words from Twitter and two from English vocabulary

3.

```
spellingSugestion("albe")
```

```
## [1] "albeit"    "alberta"   "alberto"   "albertson"
```

Four words, all form the Twitter data set (i.e. English vocabulary was checked and not used)

4.

```
spellingSugestion("hab")
```

```
## Added the following English vocabulary words: habit, habits, habitual, habitation
```

```
## [1] "habit"      "habits"     "habitual"   "habitation"
```

Only words from English vocabulary

```
spellingSugestion("q[^u]", 20) # note that you can also use regular expressions here!
```

```
##  [1] "qb"         "q3"         "q1"         "qbs"        "q2"
##  [6] "qcc"        "qkyvqdmfro" "qtr"        "qwugnniwtq" "q1nuqbgn0d"
## [11] "q2mlhmaybi" "q4"         "q4k8kmpegu" "q6bxqyggm4" "q6sicpzqzs"
## [16] "q9ctvclwi3" "qa98fk86v"  "qaarstbhr5" "qb's"       "qb1"
```

This returns all words in our vocabularies that start with `q` that is not followed by `u`. All of which are found in the Twitter vocabulary.