Problem Set 6

Overview:

In this problem set, you will be working on a shared repository with your homework group. We encourage you all to communicate with your group by creating issues on your shared repository.

We will not be asking for your git commands and output – you can simply use your command line interface. You will only need to submit (via pushing to your Github repository) your R script and any data files you create.

The purpose of this problem set will be to practice the very basics of regular expressions. The problem set will be largely based on the `stringr` cheat sheet (particularly page 2), which is also part of your required reading this week. So we suggest you read it closely! The rationale behind this problem set is that becoming comfortable referring to the regular expressions cheat sheet will improve your ability to use regular expressions to solve practical problems. This problem set is very straightforward, just practicing the fundamental skills. In the subsequent problem set, you will use regular expressions to solve practical data problems.

Part I: Fetching twitter data

Like the previous week, create a new private GitHub repository here for your group called `<team_name>_ps6` and initialize it with a `.gitignore` file. Create a new RStudio Project for this repository and complete all your work on a branch called `dev_<last_name>`.

Create a new R script called `<last_name>_script.R` where you will do all the work for this problem set.

1. Load in the following packages:
   - `tidyverse` (which will automatically load in `stringr`)
   - `rtweet`
2. In your project directory, create a new directory called `data` where you will save your data file(s). In your R script, create a `data_dir` object that stores the file path to the `data` directory. Remember to write the path relative to the project directory.
3. Like last week, you will be working with Twitter data from various UC's and affiliated COLA accounts. Create an object called `cola` that contains the following handles:

```
cola <- c("UCR4COLA", "uci4cola", "ucla4cola", "UCM4COLA", "payusmoreucb",
          "SpreadtheStrike", "UCSF4COLA", "ColaUcsd", "ucsb4cola",
          "payusmoreucsc", "ucd4cola")

cola
```

4. Use the `search_tweets()` function to search for 500 tweets from the twitter handles stored in `cola`. Save the dataframe in an object called `cola_df`.
5. Use `saveRDS()` to save your `cola_df` in a file called `<last_name>_twitter_cola.RDS` in the `data_dir`. Add this file and make a commit on your branch.
6. Subset your dataframe and call it `cola_df2` and keep only the following variables: `user_id`, `screen_name`, `created_at`, `text`.
7. Create a character vector object called `tweet_text` that is just the `text` column of `cola_df2`.

Part II: Escaping special characters in plain old strings (no regular expressions yet)

1. Use the command below to pull up the help file for "Quotes". Read the section "Character constants" in this help file, which describes and identifies **special characters**. Special characters must be "escaped" by putting a backslash \ prior to the character.

```
?"'"
```

   For each of the following special characters, create a short string (could be a phrase or sentence) that contains that special character, then print this string using both the `print()` function and the `writeLines()` function.

Special character	Represents
`\n`	newline
`\t`	tab
`\\`	backslash \
`\'`	single quote '
`\"`	double quote "

2. With respect to the previous question, explain in general why the output created by the `print()` function differs from the output created by the `writeLines()` function.

Part III: Regular expressions

Match Characters

1. Copy the following code to create the character vector `text1`. Print `text1` using the `print()` function and using the `writeLines()` function.

```
text1 <- c("abc ABC 123\t.!?\\(){}\n","abcde","aaa","bacad",".a.aa.aaa","bacad","abbaab")
```

2. For this question, the pattern to match is matches to the letter `"a"` (as in `pattern = regex("a")`).

   (A) Use `str_view_all()` to show all matches to the letter `"a"` in the character vector `text1`.
   (B) Use `str_view()` to show the first match to the letter `"a"` in each of the first 5 elements of the character vector `tweet_text` that you created in Part I.
   (C) Using pipes, first use `mutate()` and `str_detect()` to add a column named `has_match` to the dataframe `cola_df2` (no need to save a new dataframe), then use `count()` to create a frequency count of the variable `has_match`.

3. Use the `typeof()`, `class()`, `str()`, and `attributes()` functions to investigate the object `regex("a")` (no need to assign to a new object).

4. Separately for each of the following matches, repeat parts (A) (B) and (C) from question 2. This time, match to the following characters and special characters:

   - A literal period, `.`
   - A literal backslash, `\`
   - A new line, special character `\n`
   - Whitespace, special character `\s`
   - Whitespace, using POSIX special character `[:space:]`
   - A numeric digit, special character `\d`
   - A numeric digit, using POSIX special character `[:digit:]`

- Word boundaries, special character `\b`
- `.` , referring to any character except newline as opposed to a literal period

Anchors

1. Repeat questions (A) (B) and (C) from the earlier question, for the following matches:
   - String starts with a lowercase letter
   - String ends with punctuation

Quantifiers

1. Create the character vector `text2` as seen below:

```
text2 <- c(".a.aa.aaa", "abbaab")
```

2. Use `str_view_all()` to show each of the following matches and `str_count()` to count the number of matches:
   - Zero or one `"a"` in a row
   - Zero or more `"a"` in a row
   - One or more `"a"` in a row
   - Exactly 2 `"a"` in a row
   - Between 2 and 3 `"a"` in a row

Alternates

1. Next, you will be practicing with the below:

Regex	Represents
`|`	or
`[xyz]`	includes one of the characters in brackets
`[^xyz]`	anything but one of the characters in brackets

   First, create the string `text3` as seen below:

```
text3 <- "abcde"
```

2. Use `str_view_all()` to show matches to the following patterns:
   - Match to `"ab"` or `"d"`
   - Match to one of the following characters: `"a"`, `"b"`, `"e"`
   - Match to anything but one of the following characters: `"a"`, `"b"`, `"e"`
3. Using the first 10 elements of character vector `tweet_text`, use `str_view_all()` to show matches to the following patterns:
   - A hashtag symbol `"#"` or a handle symbol `"@"`
   - The text `"https"` or a handle symbol `"@"`
   - Any character that is not a vowel (uppercase or lowercase)

Putting it all together

1. You will now put everything you've learned together for creating the next few regular expressions. First, create the character vector `text4` as seen below:

```
text4 <- c("Los Angeles, CA 90024", "New York, NY 10001", "12345 Main St", "Pier 39")
```

2. Using `str_view()`, write a regular expression that will match only the 2 states.
3. Using `str_view()`, write a regular expression that will match only the 2 zip codes.
4. Using `str_view()`, write a regular expression that will match a sequence of 2 or more of any of the following characters in a row: `RSTLNE` (can be uppercase or lowercase).

Part IV: Groups, backreferences, and other `stringr` functions

1. Using `str_view()` and `text4`, write a regular expression that is able to match the first 2 elements of `text4`, which follows the form:

   `<city>, <2_letter_state_code> <5_digit_zip_code>`

   Have your regular expression contain 3 capturing groups for matching each of the city, state, and zip code parts. Save your regular expression as either a string or `regex` object in a variable called `citystatezip_regex`.
2. Using `str_subset()` and `citystatezip_regex`, filter the character vector `text4` to only include elements that matches the city, state, zip code pattern. Save this filtered vector in a variable called `locations`.
3. Using `str_match()` and `citystatezip_regex`, obtain a character matrix containing the matches for the `locations` vector and store it in a variable called `matches`. The 1st column of the matrix should contain the full match, and the next 3 columns contain matches for each capturing group.

   Create 3 objects – `cities`, `states`, and `zip_codes` – that will be character vectors containing the matched cities, states, and zip codes, respectively. (hint: assign the respective column of the `matches` matrix to each object)
4. Using `str_replace()` and `citystatezip_regex`, replace each element in `locations` from having the format:

   `<city>, <2_letter_state_code> <5_digit_zip_code>`

   To having the format:

   `<city> has a zip code <5_digit_zip_code> and is in the state <2_letter_state_code>`

Part V: I got issues

1. Navigate to the issues tab for the **rclass2** repository here.

   You can either:

   - Create a new issue posting a question you have about the class/problem set (assign instructors)
   - Answer a question that another student posted
   - Create a new issue posting about something new you learned or figured out from this class
     - If you choose this option, please mention the other members of your team and assign yourself

Paste the link to the issue you contributed to as a comment in `<last_name>_script.R`.

Please make sure to close the issue once your question has been resolved or within 1 week.

Part VI: Wrapping up

1. How much time did you spend on this problem set? Write your response as a comment in `<last_name>_script.R`.
2. Finally, add your `<last_name>_script.R` file and make a commit on your branch. Switch back to the `master` branch and merge in your `dev_<last_name>` branch. Push your work to the remote.