Problem Set 7

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 are required to do your work in branches. 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 give you more practice using regular expressions, parsing HTML from a webpage using the `rvest` package, and creating visualization(s) using `ggplot`. We will continue our example from lecture `rvest` using the https://corona.help/ link. You will use `rvest` to scrape HTML from the https://corona.help/ link, `stringr` functions to isolate information (e.g. country name, number of infected individuals), and `ggplot` to create a plot of the number of infected cases by country.

Part I: Scraping and parsing Coronavirus data

Like the previous week, create a new private GitHub repository here for your group called `<team_name>_ps7` 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` and `ggplot`)
   - `rvest`
2. In your project directory, create a new directory called `plots_<last_name>` where you will save your plot(s). In your R script, create a `plots_dir` object that stores the file path to the `plots_<last_name>` directory. Remember to write the path relative to the project directory.
3. Use `read_html()` to scrape the HTML from `https://corona.help/` and save it to a variable called `corona`. (hint: Refer to the lecture examples for this and the next few steps)
4. Use `html_nodes()` to select all rows in the "Total by country" table and save them in a variable called `corona_rows`.
5. Use `as.character()` to convert `corona_rows` to raw HTML. Save all but the header row to a variable called `rows`.
6. Use `writeLines()` and `head()` to preview the first few rows of the data.
7. Write a regular expression that will match the first two columns of each row (country name & number infected). Use capturing groups to be able to isolate these 2 pieces of information from between the HTML tags.

   Use `str_match()` to find the matches from `rows` and store the result in a variable called `corona_data`.
8. Your `corona_data` object should be a character matrix, where the 1st column is the full match, the 2nd column is the country name, and the 3rd column is the number infected.

   Create 2 objects – `country_name` and `num_infected` – that will be character vectors containing the country name and number infected, respectively. (hint: assign the respective column of the `corona_data` matrix to each object)

9. Use `str_replace_all()` to replace all the `','` in `num_infected` with an empty string. Then use `as.numeric()` to convert the vector to numeric and save the result back to `num_infected`.
10. Use `data.frame()` to create a dataframe called `corona_df` from the `country_name` and `num_infected` vectors. (hint: See lecture example here for the syntax)

Part II: Plotting Coronavirus data

1. Use `head()` to filter for the first 5 rows of `corona_df`. Then, use `geom_col()` to create a barplot with `country_name` on the x-axis and `num_infected` on the y-axis. Make sure to label the axis.

   Save the plot as `corona_top5.png` in the `plots_dir` directory.

2. Next, create the same barplot, but this time include only countries whose name either starts with a `v` or ends with a `i` (not case-sensitive). Use `filter()` and `str_detect()` to help subset your dataframe.

   Save the plot as `corona_subset.png` in the `plots_dir` directory.

   You can use this syntax to save the plots:

   ```
   png(file.path(plots_dir, 'plot_name.png'))
   dev.off()
   ```

3. Add both plots and commit with the message `Coronavirus plots`.
4. Push your `dev_<last_name>` branch to the remote. Remember you'll need to set the upstream branch during this initial push.
5. On GitHub, create a pull request to have the changes from your `dev_<last_name>` branch merged to `master`. Assign one of your group members as the reviewer. Do not assign the person who has assigned you as their reviewer, so that everyone will get a chance to review for someone else.
6. When you are assigned as reviewer by one of your group members, navigate to their pull request and take a look at their plot.

   Under the `Files` tab, click on `Review changes` and select the `Request changes` option. Leave a comment stating one thing you'd like to see changed about either plot, then click `Submit review`.

7. Once you receive feedback on your own plot, make the requested changes and push your revised plot to the branch. Re-request a review from your reviewer.
8. Finally, once you receive a re-request for your review on your group member's plot, make sure to approve their new changes, then click on `Merge pull request` to merge the changes into `master`.

Part III: 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, once your pull request has been approved and merged to `master` by your group member, you can add your `<last_name>_script.R` file to the `dev_<last_name>` branch and make a commit. Switch back to the `master` branch and merge in your `dev_<last_name>` branch. Push your work to the remote.