Econ 5253 - Spring 2022 **Problem Set 4** **Due: Feb. 22 beginning of class**

This problem set will provide an opportunity for you to continue practicing with the command line and executing batch jobs on the OSCER cluster. You will also get practice importing data and working in Spark.

As with the previous problem sets, you will submit this problem set by pushing the document to *your* (private) fork of the class repository. You will put this and all other problem sets in the path `/DScourseS22/ProblemSets/PS4/` and name the file `PS4_LastName.*`. Your OSCER home directory and GitHub repository should be perfectly in sync, such that I should be able to find these materials by looking in either place. Your directory should contain four files:

- `PS4a_LastName.R` (first R exercise; though you can also do this in Python or Julia if you prefer)
- `PS4b_LastName.R` (`sparkR` exercise)
- `PS4_LastName.tex`
- `PS4_LastName.pdf`

1. Log in to OSCER, change to the directory where you cloned your forked GitHub repository (probably `~/DScourseS22`), and make sure the OSCER version of your repository is synchronized with what is listed on GitHub by issuing a pull. That is, type `git pull origin master` from your OSCER `DScourseS22` folder.
2. Synchronize your fork with the class repository by doing a `git pull upstream master`.
    - Before doing this, make sure that you have set your default git text editor to Nano (and not Vim) by typing the following at the command line: `git config --global core.editor "nano"`

Making your SLURM job scripts visible from any directory

3. In class last week, you practiced running simple R or Python scripts on the OSCER cluster using the `Rbatch`, `Pythonbatch`, and `juliabatch` scripts located in the `SLURM/` folder of our course GitHub repository. Recall that the syntax for these commands was (assuming you are in the `SLURM/` directory): `./Rbatch rscript.R rscriptoutput.log 1:00 my-email@address.com`, where the "`1:00`" argument is a number indicating how long the job should run for.

    Now, I'd like you to move these files to a place in your OSCER directory tree where they can be executed from *any* folder (not just the `SLURM/` folder). To do so, follow these steps:

1. Change to your home directory: `cd ~`
2. Create a new directory called `bin/` by typing `mkdir bin`
3. Copy the `*batch` files from your `SLURM/` folder to the `~/bin/` folder using `cp`.
4. Change to the `bin/` folder and do a listing and make sure that the files copied successfully, and that they are executable (the filenames should be colored green).[1]
5. Go back to your home folder (`cd ~`) and type `which Rbatch`. It should return with `~/bin/Rbatch`. Now you can execute the `Rbatch` script from wherever you are on OSCER![2]
   (a) Note that, when executing these scripts from now on, you don't need to prepend them with "`./`" because "`./`" is telling Linux to execute the file that's in the current directory. So in the future, execute these scripts by simply typing `Rbatch myfile.R` and **not** `./Rbatch myfile.R`.

Making Spark executables visible from any directory

4. This follows a bit on the previous question. What you will now do is edit your `~/.bash_profile` file to make it so you can simply type `sparkR` or `pyspark` to automatically open the Spark API of your choice.

   To do this, open in `nano` the `.bash_profile` file which is located in your home directory.

   Near the bottom of the file, you should see the phrase `EXPORT PATH`. Just above this line, type `module load Spark/2.0.0`. Save and close the file, and then log out of OSCER.

   Once you've logged back in to OSCER, verify that your modification worked by typing `which sparkR` at the command line. The command prompt should reply with a long file path.

   Type `sparkR` at the command line and you should be able to use Spark's R API.

Practice with JSON files (R exercise part 1)

5. This question will help you get comfortable working with (and converting from) JSON data, which is the most common data format for APIs that house web data.

[1] If they are not green, issue a `chmod 774 filename` command on each file.

[2] For those curious about what's going on "under the hood," there is a Linux variable called `$PATH` which tells the system where to look for executable files. This `$PATH` variable is loaded whenever you log in because it is contained in the file `~/.bash_profile`. By making changes to your `.bash_profile` file, you can change your login envrionment without having to repeat commands every time you log in.

(a) Download the following file from within R, Python, or Julia: `"https://www.vizgr.org/historical-events/search.php?format=json&begin_date=00000101&end_date=20220219&lang=en"`

This website lists historical events from Jan 1, 0000 until Feb 19, 2022.

The way to do this is to call `wget` (which is a system command) from inside R/Python/Julia. Note that we want to specify the local name of this file (call it `dates.json`). To do that, we say `wget -O filename.extension "urlpath"` (note: that's a letter O, not a number 0; also pay attention to the quotation marks).

- R sytnax is: `system('linux shell command')`
- Python sytnax is: `call(["linux", "shell", "command"])`[3]
- Julia sytnax is: `run(`linux shell command`)`

(b) Now print your file to the console by typing `cat dates.json` (or whatever you choose to name the file) within the system call.

(c) This file is ugly, so let's make it a little easier to deal with by converting it to a data frame.

- If you use R, you will need to call the libraries `jsonlite` and `tidyverse`. You may need to install them first. The code to convert to a dataframe requires two steps. First, convert the JSON to a list: `mylist <- fromJSON('dates.json')`. (Make sure you call the file by whatever you called it in part (b).) Second, convert the list to a data frame (and remove the first element, since in this case it is not useful): `mydf <- bind_rows(mylist$result[-1])`

(d) Check what type of object `mydf` is. What type of an object is `mydf$date`?

- In R, this is done with `class()`.
- In Python, this is `class()`.
- In Julia, this is `typeof()`.

(e) List the first n rows of the `mydf` dataframe.

(f) Put all of these commands into an R, Python, or Julia script and then run it from your `PS4/` directory using `Rbatch`, `Pythonbatch`, or `juliabatch`. Remember the correct syntax which is listed in Question 3 of this homework.

What I wanted you to take away from this exercise is that there is no one-to-one mapping from JSON/YAML files to tabular data. So creating a tabular data frame

[3] This requires the `call` function from the `import` library. Also note that spaces in the command need to be in separate strings.

from a JSON requires a little extra work. The same holds true for other data types like XML and HTML (though these may be closer to a one-to-one tabular representation).

Also, note that the `fromJSON` and other functions can accept a URL as an argument. I had you use the shell just so you can get comfortable with accessing the shell from within R/Julia/Python.

Practice with sparklyr (R exercise part 2)

6. This exercise will familiarize yourself with `sparklyr` which is how one can use Spark through R. The walkthrough that I am giving you can also be found at https://spark.rstudio.com/. Please create an R script called `PS4b_LastName.R` which contains all of your `sparklyr` commands (so that you could easily reproduce your work whenever called upon).

    1. Open an R session on OSCER by typing `R` at the command prompt.
    2. Make sure you have installed the `sparklyr` and `tidyverse` packages.
    3. Load `sparklyr` and `tidyverse` packages.
    4. Set up a connection to Spark by issuing the following commands:

        ```
        spark_install(version = "3.0.0")
        sc <- spark_connect(master = "local")
        ```

    5. Create a tibble called `df1` that loads in the `iris` data.[4]
    6. Now copy this tibble into Spark, calling it `df`. The command for this is `df <- copy_to(sc, df1)`.
    7. Verify that the two dataframe are different types: type `class(df1)` and `class(df)`. What is the class of each?
    8. Are the column names any different across the two objects? If so, why might that be?
    9. Next, we will apply the common RDD/SQL operation: `select`
        (a) List the first 6 rows of the `Sepal_Length` and `Species` columns of `df`. This can be done by typing `df %>% select(Sepal_Length,Species) %>% head %>% print`.
    10. Now let's do another common RDD operation: `filter`

[4] Hint: use the command `as_tibble()`.

(a) List the first 6 rows of all columns of `df` where `Sepal_Length` is larger than 5.5. This can be done by typing `df %>% filter(Sepal_Length>5.5) %>% head %>% print`.

11. Combine the two previous exercises into one line (that is, put both the `select` and `filter` operations into one line using the `dplyr` pipeline.

12. Another useful RDD operation is "group_by." We can compute the average sepal length, as well as the number of observations, by each of the three iris species: `df2 <- df %>% group_by(Species) %>% summarize(mean = mean(Sepal_Length), count = n()) %>% head %>% print`.

13. Finally, a common RDD operation is to sort. We can sort (`arrange()`) the above "grouped by" RDD by any of the three variables it contains.

    (a) Re-execute the previous call, this time assigning `df2` to the output.

    (b) Now use the `arrange()` function to sort the result ascending by species name: `df2 %>% arrange(Species) %>% head %>% print`

7. Go to `www.overleaf.com` and create another .tex document, this time naming it `PS4_LastName.tex`. In it, tell me about some data sources that you would be interested in scraping from. These could be, for example: classical texts from Project Gutenberg, tweets that include a particular hashtag, college or professional sports statistics, financial market data, etc. For anything you are interested in, there is almost surely data that is freely available on the internet, and most data sources come with highly accessible APIs for R or Python.

   In another part of your .tex file, answer the questions raised in the various parts of the previous question.

8. Compile your .tex file, download the PDF and .tex file, and transfer it to your cloned repository on OSCER using your SFTP client of choice (or via `scp` from your laptop terminal). You may also copy and paste your .tex file from your browser directly into your terminal via `nano` if you prefer, but you will need to use SFTP or `scp` to transer the PDF.[5]

9. You should turn in the following files: .tex, .pdf, and two .R scripts. Make sure that these files each have the correct naming convention (see top of this problem set for directions) and are located in the correct directory (i.e. `~/DScourseS22/ProblemSets/PS4`).

[5] If you want to try out something new, you can compile your .tex file on OSCER by typing `pdflatex myfile.tex` at the command prompt of the appropriate directory. This will create the PDF directly on OSCER, removing the requirement to use SFTP or `scp` to move the file over.

10. Synchronize your local git repository (in your OSCER home directory) with your GitHub fork by using the commands in Problem Set 2 (i.e. `git add`, `git commit -m "message"`, and `git push origin master`). More simply, you may also just go to your fork on GitHub and click the button that says "Fetch upstream." Then make sure to pull any changes to your local copy of the fork. Once you have done this, issue a `git pull` from the location of your other local git repository (e.g. on your personal computer). Verify that the PS4 files appear in the appropriate place in your other local repository.