Lab 1: JSON Manipulation

Due date: April 13, 10:00am.

Note: Lab 2 assignment may come out on Friday, April 10.

Lab Assignment

Assignment Preparation

This is an individual lab.

In this lab you will review how to work with JSON documents in two programming languages: Python and Java. We will need to develop software in both languages that works with JSON throughout the course; this assignment allows you to take care of the boring details.

Additionally, this assignment introduces the types of tasks we will be working on solving using distributed computing techniques throughout the quarter. You will implement each of these tasks at least once more — and possibly - up to three more times during the quarter using different distributed computing frameworks.

DISCLAIMER

This lab has you work with **current data** about COVID-19 infections in the United States. This includes information about both infections, and deaths due to COVID-19. Some of the assignments have you analyze the fatalities due to COVID-19 and compare them across time, and across different states. I am giving you this assignment, because I want to emphasize the importance of certain types of data analysis. In doing so, I understand fully that this may be a sensitive issue to some of you, if you have experienced cases of COVID-19 infections or deaths among the members of your family and/or friends.

I hope that all of you can see the importance of timely analysis of COVID-19 data (even if some of the analytical tasks I am asking you to implement are somewhat contrived). If, however, you prefer to work with different data, let me know. I have a prior year assignment that uses a different dataset, that I can provide for you in lieu of this assignment. The two assignments meet the same learning objectives and test the same skills.

Dataset

Given the circumstances, we will be using the information about COVID-19 infections in the United States. The data comes from the COVID Tracking Project web site, specifically, from this URL:

https://covidtracking.com/api

We will be using the JSON version of the daily US States data, available directly at this endpoint:

https://covidtracking.com/api/states/daily

For the sake of reproducibility, we will be using the data file I downloaded on April 5, that includes all available data from the beginning of the tracking (March 3, 2020) through April 5, 2020. The data file is available for download from the course web site.

The COVID Tracking project Website describes the format of each JSON object in the collection as follows:

- state - State or territory postal code abbreviation.
- positive - Total cumulative positive test results.
- positiveIncrease - Increase from the day before.
- negative - Total cumulative negative test results.
- negativeIncrease - Increase from the day before.
- pending - Tests that have been submitted to a lab but no results have been reported yet.
- totalTestResults - Calculated value (positive + negative) of total test results.
- totalTestResultsIncrease - Increase from the day before.
- hospitalized - Total cumulative number of people hospitalized.
- hospitalizedIncrease - Increase from the day before.
- death - Total cumulative number of people that have died.
- deathIncrease - Increase from the day before.

- dateChecked - ISO 8601 date of the time we saved visited their website
- total - DEPRECATED Will be removed in the future. (positive + negative + pending). Pending has been an unstable value and should not count in any totals.

In addition to these attributes, the JSON objects will contain the following attributes (explained elsewhere in the API documentation):

- date - date for which the data is provided in the YYYYMMDD format (note: JSON treats this value as a number - make sure you parse correctly).
- fips - Federal Information Processing Standard state code
- hash - the hash code of the record
- hospitalizedCurrently - number of people currently hospitalized
- hospitalizedCumulative - appears to be the new name for the hospitalized attribute
- inIcuCurrently - number of people currently in the ICU
- inIcuCumulative - total cumulative number of people who required ICU hospitalization
- onVentilatorCurrently - number of people currently on the ventilator
- onVentilatorCumulative - total cumulative number of people who at some point were on ventilator
- recovered - total cumulative number of people who recovered from COVID-19

Note: "DEPRECATED" attribute means an attribute that can be found in some of the earlier JSON records, that that is not found in the most recent ones.

An example JSON object (pretty printed) is provided below:

```
{ "date":20200405,
  "state":"MD",
  "positive":3609,
  "negative":24728,
  "pending":null,
  "hospitalizedCurrently":null,
  "hospitalizedCumulative":936,
  "inIcuCurrently":null,
  "inIcuCumulative":null,
  "onVentilatorCurrently":null,
  "onVentilatorCumulative":null,
  "recovered":159,
  "hash":"cac549d0b74daccf3ab34404fc5dd3812b44bbff",
  "dateChecked":"2020-04-05T20:00:00Z",
  "death":67,"hospitalized":936,
  "total":28337,
  "totalTestResults":28337,
  "posNeg":28337,"fips":"24",
  "deathIncrease":14,
  "hospitalizedIncrease":115,
  "negativeIncrease":2243,
  "positiveIncrease":484,
  "totalTestResultsIncrease":2727}
```

In your processing of this data you will have to take into account the fact that different JSON objects in the collection can have different format – earlier objects will have fewer attributes; latter objects will have more attributes, and will have some attributes renamed for clarity. The dataset has NULL values, and you will be responsible for properly handling those as well.

The data is provided to you in a single JSON file called `daily.json`. Note that this file is a proper JSON object itself, i.e., it is a JSON array of JSON objects. This makes automated parsing straightforward.

Python Assignment

You will write four Python programs. Each program will take `daily.json` file as input. Each program shall print all output to screen (if we need to capture it in a file, we will redirect the output). Some programs will output a JSON array. Some programs will output data in human-readable format – each program description will provide individual output description.

Please follow the strict naming conventions provided below. Your code will be run automatically. Failure of the automated scripts to run your code is grounds for deduction of points for non-compliance with assignment specifications.

transformCOVID.py: This program shall output an array of JSON objects - one per state. Each JSON object shall have the following format:

- state - state abbreviation (same as state in the input file)
- startDate - the first date on which information about positive cases is present in the input. Note: this may not be the date of the first case in the state - that may have been earlier than March 3. It is simply the date starting with which meaningful data about the state is available.
- newCases - JSON array containing the list of the numbers of new cases for each day, *starting with the startDate*. Any `null` values in the original data can be replaced with 0 for the purpose of this assignment (even if it is strictly not correct as far as the actual accounting is concerned).

- newDeaths - JSON array containing the list of numbers of new deaths for each day, *starting with the startDate*. Similarly, you can replace null values with zeros for the purpose of this assignment.

- mortalityRates - JSON array containing the list of mortality rates for each day, *starting with the startDate*. The mortality rate is computed as
$$\frac{\mathsf{TotalCumulativeDeaths}}{\mathsf{TotalCumulativePositiveCases}}$$

You can truncate the mortality rate to four decimal points. You can also represent it as a percentage in the format $XX.YY$.

highestRatio.py: For this exercise you will, for each state, find the date on which it had the *highest ratio of the number of total cumulative positive cases to the number of total cumulative negative casses.* The output for this program shall be a collection of CSV (comma-separated values) rows, with each row representing a single state, and containing the following columns:

1. state – the state abbreviation
2. date – the date of the highest ratio
3. totalPositive – the total cumulative number of positive cases on that day
4. totalNegative – the total cumulative number of negative tests on that day
5. ratio – the ratio of totalPositive and totalNegative. Can be formatted to three decimal places.

Output the states in alphabetical order of the state code.

NOTE: your output may contain more than 50 rows, because the dataset includes data for Washington, DC, as well as other US territories and possessions.

moreThanCA.py: We want to find the day on which the total number of states with a higher overall mortality rate due to COVID-19 was the highest. We compute the mortality rate for a state on a given day as the percent of all total cumulative positive cases that ended in a fatality. Note that due to lack of data, this information may not be available for each state and each day in the database.

Consider only the days on which the mortality rate for California can be computed. For a given date, ignore any state for which you cannot compute the mortality rate due to unavailability of appropriate data in the JSON object (or lack of the JSON object itself for the state and the date). Compare the mortality rates for each *other state* (including territories and possessions) to California's for each given day, and tally the number of states that exceeded California's rate on each day. Find the day where the largest number of states exceeded California's mortality rate.

Report the following infromation (just print). First, output the date. Second, output the list of states whose mortality rates exceeded California's mortality rate on that date, in decreasing order of the mortality rates, one state per line of output, *ending the list with California itself.* For each state, report its name (abbreviation), and the computed mortality rate. The mortality rate can be formatted either as a percentage in the XX.YY% format, or as a number in the $[0, 1]$ range with four decimal places.

grimLeaderboard.py: For each date found in the dataset output the state with the largest number of total cumulative positive cases. Print output in the CSV format, with each line represening a single date in chronological order (starting with the earliest date), and containing the date, the abbreviation of the state, and the total cumulative number of cases. In case of ties, report the state that is the first lexicographically (using the state abbreviations). E.g., "CA" lexicographically precedes "NY", therefore any ties between these two states will result in reporting "CA".

Java Assignment

Your Java assignment is similar to your Python assignment. You will write two programs.

stateInfo.java: For each state report the following information in JSON format:

- state – state abbreviation
- daysSince100 – number of days since the total cumulative number of cases hit or exceeded 100 (if the total cumulative number of cases exceeded 100 on March 3, then report the total number of days since March 3)
- averageCases – average number of new cases per day since the day the total cumulative number of cases hit or exceeded 100 (following the same caveat as daysSince100 attribute).
- dailyNewCases – JSON array containing the list of daily new case numbers in the state starting on the day when the total cumulative number of cases hit or exceeded 100 (following the same caveat as daysSince100 attribute).
- dailyCoeff – JSON array containing the list of daily growth coefficients starting on the day when the total cumulative number of cases hit or exceeded 100 (following the same caveat as daysSince100 attribute).

The growth coefficient is the ratio between the number of new positive cases on a given day, and a number of new positive cases on the day before. Report growth coefficient of 0 if you are unable to compute it (null values).

correlation.java: This is the most involved (if not the most complex) task. We will compute the correlation between the number of positive cases and the number of deaths for each state. However, we will do this with a "shift". The procedure is described below.

For a given state, we need to construct two vectors: a vector of cumulative COVID-19 cases, and a vector of deaths due to COVID-19.

First, we construct the vector of the numbers of deaths due to COVID-19. We skip all dates from the beginning of the dataset until our data shows a non-zero number of cumulative deaths. After this, we include in our vector, all cumulative number of deaths values until April 5.

Second, we constructs the vector of positive COVID-19 cases in a state. This vector is constructed in two steps. First, we find the date of the first non-zero cumulative number of positive cCOVID-19 cases reported. We then construct a vector of all numbers of total positive cases from the abovementioned date and through April 5.

The two vectors we constructed though will be of different length – we expect the vector of positive COVID-19 cases to be longer (positive cases are reported ahead of deaths typically). To make both vectors of be of the same size we remove from the longer vector (the vector of positive COVID-19 cases in a state) the necessary number of values from the end of the vector.

Example. Consider the following two lists. The list of cumulative number of deaths is $0, 0, 0, 0, 0, 1, 3, 5, 8, 10, 16, 23$. The list of positive COVID-19 cases is $0, 0, 1, 1, 4, 8, 14, 19, 29, 41, 48, 63$. The extracted vector of deaths is $(1, 3, 5, 8, 10, 16, 23)$ and has the length of 7. The extracted vector of positive COVID-19 cases is $(1, 1, 4, 8, 14, 19, 29, 41, 48, 63)$ and has 10 elements. We crop the last three values in the vector. Our final vectors, alinged against each other are:

```
fatalities: (1, 3, 5, 8, 10, 16, 23)
infections: (1, 1, 4, 8, 14, 19, 29)
```

(do not be concerned about number of deaths possibly exceeding number of infections in this alingment - we know that those happened on different dates).

With the two vectors determined this way, we compute Pearson's correlation. Pearson's correlation between two vectors of values $\mathbf{a} = (a_1, \ldots, a_n)$ and $\mathbf{b} = (b_1, \ldots, b_n)$ is computed using the following formula:

$$pearson = \frac{\sum_{i=1}^{n}(a_i - \mu_a)(b_i - \mu_b)}{\sqrt{\sum_{i=1}^{n}(a_i - \mu_a)^2}\sqrt{\sum_{i=1}^{n}(b_i - \mu_b)^2}},$$

where

$$\mu_a = \frac{1}{n}\sum_{i=1}^{n} a_i; \mu_b \frac{1}{n}\sum_{i=1}^{n}$$

are the means of vectors **a** and **b** respectively.

Your goal: for the `correlation.java` program, for each state compute the correlation between the fatalities due to COVID-19 and cumulative numbers of infections as described above. Report all correlations in the alphabetical order of the state abbreviation. You output shall be in CSV format.

If a given state does not provide enough data to compute the correlation, ignore it, and do not include it in the output.

General Comments and submission

All your programs, both written in Python and in Java must produce valid JSON as output when asked for. This means that when your programs are producing multiple JSON documents as output, the output shall be a JSON array. For a single JSON object, there is no need to encapsulate it in an array. For CSV-style outputs, feel free to output the header line with column names first, followed by the actual output, but this is left to you as an option.

All your programs shall take a file name of the input file as an input parameter.

I tried to predict some of the situations when data quality may be of issue, but there may be additional scenarios where lack of data may interfere with your computations. In each case, please handle the data appropriately. Your programs should not crash on ”inconvenient” data. Rather - if you cannot compute something for a given state/date combination - ignore it and do not include in your output.

There are common operations your programs may wind up performing. I recommend developing a Python library of helper functions shared among your four Python programs. You can submit it together with your other files. With Java - there is less of a need for a common framework (only two tasks, and they are more disparate), but if you want to build a class/interface to work with both programs in this assignment, you are welcome to do so.

Each program/README file you submit must contain your name and cal poly email address.

Submit the six programs you developed, and supplemental files (e.g., your Python library) and a `README` file containing any additional information I may need to know to compile and run your programs (e.g., the `javac` compilation command, if it incorporates some parameters I may not be immediately aware of).

You can develop Python code in Jupyter notebooks, but for this assignment, I want you to submit standalone Python programs.

When testing your code, both Python and Java, assume that I will run it on `unix3.csc.calpoly.edu`.

Use Python 3 syntax.

Use `handin` on `unixN` (e.g., `unix3.csc.calpoly.edu`) servers to submit as follows:

```
$ handin dekhtyar lab01 <FILES>
```

Note: later we may move to `handin` on the `ambari-head.csc.calpoly.edu`, but `Lab 1` programs do not require the use of the cluster, so we will grade them on the CSL servers.

Good Luck!