UE20CS322 Big Data Assignment 3

Working with Spark and Kafka

This is the third assignment for the UE20CS322 Big Data course. The assignment consists of two tasks. The first task focuses on working with Apache Spark and the second task focuses on working with Apache Kafka.

The files required for the assignment can be found here.

Assignment Objectives and Outcomes

1. Get familiar with Apache Spark and learn about the speed and efficiency of Spark.
2. Work with Apache Kafka and understand how a Producer - Consumer model works.

Ethical Practices

Please submit original code only. You can discuss your approach with your friends but you must write original code. All solutions must be submied through the portal. We will perform a plagiarism check on the code and **you will be penalized if your code is found to be plagiarized.**

Submission Deadline

10th November 2022, 11:59 PM

The portal uptimes will be communicated to you in due course. Please do not wait till the last minute to submit your code since wait times may be long.

Submission Link

Portal for Big Data RR Campus Assignment Submissions

Portal for Big Data EC Campus Assignment Submissions

Submission Guidelines

You will need to make the following changes to your Python scripts before submitting them.

1. Include the following shebang at the top of your Python scripts.

```
#!/usr/bin/env python3
```

2. Convert your files to executable using the following command.

```
chmod +x *.py
```

3. Convert line breaks in DOS format to Unix format (this is necessary if you are coding on Windows without which your code will not run our portal).

```
dos2unix *.py
```

Software/Languages to be used:

1. Python `3.10.x`
2. Hadoop `v3.3.3`
3. Apache Spark `v3.3.0`
4. Apache Kafka `v3.3.1`

Additionally, the following Python libraries are required:

1. `pyspark==3.3.0`
2. `kafka-python==2.0.2`

No other libraries are allowed.

Task 1 - Spark

Dataset

You will be working with a subset of the simplified version (reduced columns) of the Los Angeles Parking Citations dataset compiled by the authorities of the City of Los Angeles. You can find the required files here. The columns in the dataset are as follows:

1. `RP State Plate` - The state in which the vehicle was registered.
2. `Ticket number` - Ticket number issued by the officer.
3. `Issue Date` - The date on which the ticket was issued.
4. `Color` - Color of the vehicle.
5. `Location` - The location where the violation occurred.
6. `Violation code` - The violation code.
7. `Fine amount` - The fine amount.

Problem Statement

Find the Ticket numbers of citations for a **white car** where the fine is greater than the average fine of ALL cars registered to the state.

Description

Use the spark-5-percent-dataset.csv file to find the ticket numbers of traffic violators with a white car who have been issued a ticket where the Fine Amount turned out to be greater than the average Fine Amount of the state in which the vehicle was registered.

Input data characteristics: The table should be deduplicated on column `Ticket number` which uniquely identifies every citation. The vehicle's state is given in the RP State Plate column. Color code for white is WH. You may drop any rows with NA values in any of the columns.

Output file requirements: The output should be a CSV file with one Ticket number per line. Do not write the header in the output file. Coalesce the output to a single file.

How to run: You are required to load the dataset into a Spark RDD or DataFrame and perform the required operations on it. You can use the `spark-submit` command to run your code. The command should be as follows:

```
$SPARKHOME/bin/spark-submit spark-solution.py <input_file> <output_folder>
```

Important!!

1. The input dataset filename and output solution folder name should not be hardcoded. The filenames should be passed as command line arguments.
2. Loops are not allowed. You must use Spark's RDDs and DataFrames to solve the problem statement.
3. The filename of the code should be `spark-solution.py` .
4. Usage of direct file interaction commands such as python's `open()` is not allowed. You must use Spark's RDDs and DataFrames to solve the problem statement.
5. Make sure to always fetch the available `SparkContext` / `SparkSession` object using the `getOrCreate()` method instead of creating a new one. This will prevent any errors while attempting to create a new SparkContext/SparkSession object to connect to the Spark cluster.
6. **Sort the output in ascending order of Ticket number.**

Sample Input and Output

The sample input file is provided here. The sample output file is provided here. You can use the `diff` command to check if your output is correct.

```
diff spark-5-percent-dataset-solution.csv your-solution.csv
```

Task 2 - Kafka

Dataset

You will be working with the Indian Agriculture Data to help the Farmers, Value Chain, and the Economy . We have slightly modified the dataset. You can find the required files here. The columns in the dataset are as follows:

1. `State` - State in which the crop was grown.
2. `District` - District in which the crop was grown.
3. `Market` - Market in which the crop was sold.
4. `Commodity` - Classification of the crop grown into different commodities.
5. `Variety` - Variety the crop falls under.
6. `Arrival Date` - Arrival date of the crop.
7. `Min Price` - Minimum price of the crop.
8. `Max Price` - Maximum price of the crop.
9. `Modal Price` - Modal price of the crop.

Problem Statement

Find the average minimum and maximum prices of the crops grown in each state.

Description

Use the kafka-5-percent-dataset.csv file to find the average minimum and maximum prices of the crops grown in each state. You are required to use Kafka's producer and consumer APIs to solve the problem statement. The output should be in the following format:

```
{
    "State1": {
        "Min": 100,
        "Max": 200
    },
    "State2": {
        "Min": 200,
        "Max": 300
    }
}
```

`State1` and `State2` are the states in which the crops were grown and `Min` and `Max` are the average minimum and maximum prices of the crops grown in each state respectively.

The logic for the producer and consumer is up to you. The input to the producer file will be streamed through the standard input.

You are required to use the `kafka-python` library to solve the problem statement. You should have two files, one for the producer and one for the consumer. The producer should be named `kafka-producer.py` and the consumer should be named `kafka-consumer.py`.

To test your code, run the consumer file first and then the producer file in a separate terminal.

```
python3 kafka-consumer.py topicName > output.json
```

```
cat kafka-5-percent-dataset.csv | python3 kafka-producer.py topicName
```

Important!!

1. The topic name should not be hardcoded. The topic name should be passed as a command line argument for both the producer and consumer files.
2. There is a special line in the end of the input file named `EOF`. This is done so that the consumer knows when to stop reading from the topic. You must not include this line in the output.
3. Usage of direct file interaction commands such as python's `open()` is not allowed. You must use Kafka's producer and consumer APIs to solve the problem statement.
4. Round off the average minimum and maximum prices to two decimal places.
5. **Sort the output in ascending order of the state names.**
6. Print the `json.dumps()` of the dictionary to the standard output and not the dictionary itself.

Sample Input and Output

The sample input file is provided here. The sample output file is provided here. You can use the `diff` command to check if your output is correct.

```
diff <(jq --sort-keys . output.json) <(jq --sort-keys . kafka-5-percent-dataset-solution.json)
```