Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

BIG DATA & ANALYTICS

ASSIGNMENT 2: SPARK SQL & SPARK STRUCTURED STREAMING

BACKGROUND.

It's been already a few weeks since you started your short-term internship in the Data Analytics Department of the start-up OptimiseYourJourney, which will enter the market next year with a clear goal in mind: "*leverage Big Data technologies for improving the user experience in transportation*". Your contribution in Assignment 1 has proven the potential OptimiseYourJourney can obtain by applying MapReduce to analyse large-scale public transportation datasets as the one in the New York City Bike Sharing System: https://www.citibikenyc.com/

OptimiseYourJourney

In the department meeting that has just finished your boss was particularly happy, again.

- The very same dataset from Assignment 1 (let's call it my_dataset_1) provides an opportunity to leverage other large-scale data analysis libraries, such as Spark SQL.
- The streaming of a subset of the dataset allows you to explore the potential of Spark Structured Streaming, a library of Spark specialised in real-time data analysis.

<u>DATASET 1:</u> (folder my_dataset)

This dataset occupies ~80MB and contains 73 files. Each file contains all the trips registered the CitiBike system for a concrete day:

- 2019_05_01.csv => All trips registered on the 1st of May of 2019.
- 2019_05_02.csv => All trips registered on the 2nd of May of 2019.
- ...
- 2019_07_12.csv => All trips registered on the 12th of July of 2019.

Altogether, the files contain 444,110 rows. Each row contains the following fields: ***start_time , stop_time , trip_duration , start_station_id , start_station_name , start_station_latitude , start_station_longitude , stop_station_id , stop_station_name , stop_station_latitude , stop_station_longitude , bike_id , user_type , birth_year , gender , trip_id***

- **(00)** ***start_time***
  - A String representing the time the trip started at.
    <%Y/%m/%d %H:%M:%S>
  - Example: "2019/05/02 10:05:00"
- **(01)** ***stop_time***
  - A String representing the time the trip finished at.
    <%Y/%m/%d %H:%M:%S>
  - Example: "2019/05/02 10:10:00"
- **(02)** ***trip_duration***
  - An Integer representing the duration of the trip.
  - Example: 300
- **(03)** ***start_station_id***
  - An Integer representing the ID of the CityBike station the trip started from.
  - Example: 150
- **(04)** ***start_station_name***
  - A String representing the name of the CitiBike station the trip started from.
  - Example: "E 2 St &; Avenue C".
- **(05)** ***start_station_latitude***
  - A Float representing the latitude of the CitiBike station the trip started from.
  - Example: 40.7208736
- **(06)** ***start_station_longitude***
  - A Float representing the longitude of the CitiBike station the trip started from.
  - Example: -73.98085795
- **(07)** ***stop_station_id***
  - An Integer representing the ID of the CityBike station the trip stopped at.
  - Example: 150
- **(08)** ***stop_station_name***
  - A String representing the name of the CitiBike station the trip stopped at.
  - Example: "E 2 St &; Avenue C".

- **(09)** ***stop_station_latitude***
    - A Float representing the latitude of the CitiBike station the trip stopped at.
    - Example: 40.7208736
- **(10)** ***stop_station_longitude***
    - A Float representing the longitude of the CitiBike station the trip stopped at.
    - Example: -73.98085795
- **(11)** ***bike_id***
    - An Integer representing the id of the bike used in the trip.
    - Example: 33882.
- **(12)** ***user_type***
    - A String representing the type of user using the bike (it can be either "Subscriber" or "Customer").
    - Example: "Subscriber".
- **(13)** ***birth_year***
    - An Integer representing the birth year of the user using the bike.
    - Example: 1990.
- **(14)** ***gender***
    - An Integer representing the gender of the user using the bike (it can be either 0 => Unknown; 1 => male; 2 => female).
    - Example: 2.
- **(15)** ***trip_id***
    - An Integer representing the id of the trip.
    - Example: 190.

DATASET 2: (folder my_streaming_dataset)

This dataset contains a subset of Dataset 1, consisting of all trips of May 1st 2019, from 10am to 2pm, with the trips sorted by chronological order and split among 1h blocks:

- **file1.csv** ➔ All trips on 2019/05/01 in the time interval [10am, 11am)
- **file2.csv** ➔ All trips on 2019/05/01 in the time interval [11am, 12pm)
- **file3.csv** ➔ All trips on 2019/05/01 in the time interval [12pm, 1pm)
- **file4.csv** ➔ All trips on 2019/05/01 in the time interval [1pm, 2pm)

The dataset is used to simulate the *real-time* streaming of the trips via the 1h batches files. In particular, a new file of Dataset 2 is received every time_step_interval seconds in the folder my_monitoring, for its further data processing.

MY_CODE

This folder **my_code** contains 6 subfolders ***A02_Part?***, one folder per exercise.

Each folder ***A02_Part?*** contains the file ***A02_Part?.py***

This is the file specifying the exercise. You must complete it.

MY_RESULTS

This folder **my_results** contains the following subfolders:

- **Assignment_Solutions**:

  This is the folder with the expected solutions to the exercises.

- **Student_Solutions**:

  This is the folder with the actual solutions to the exercises (based on what you have programmed).

- **my_A02_check_results.py**:

  This is a program you can use to ensure your exercises in Parts 1, 2 and 3 are indeed producing the expected result. To do so, modify the line 74 of the program with the exercise you want to test (e.g., value 2 to test the exercise 2) and run the program. If it prints "True" means that you have passed the test; otherwise "False" means the result of your query differs from the expected one.

- **my_A02_Part4_check_results.py**:

  This is a program you can use to ensure you exercise in Part 4 is indeed producing the expected result. To do so, run the program. If it prints "True" means that you have passed the test; otherwise "False" means the result of your query differs from the expected one.

Main Message

Use the programs **my_A02_check_results.py** and **my_A02_Part4_check_results.py** to ensure that all your Spark SQL and Spark Structured Streaming exercises produce the expected output (and in the right format!).

TASKS / EXERCISES.

The tasks / exercises to be completed as part of the assignment are described in the next pages:

- The following exercises are placed in the folder **my_code:**
  1. **A02_Part1**/A02_Part1.py
  2. **A02_Part2**/A02_Part2.py
  3. **A02_Part3**/A02_Part3.py
  4. **A02_Part4**/A02_Part4.py

  Marks are as follows:
  1. **A02_Part1**/A02_Part1.py => 25 marks
  2. **A02_Part2**/A02_Part2.py => 25 marks
  3. **A02_Part3**/A02_Part3.py => 25 marks
  4. **A02_Part4**/A02_Part4.py => 25 marks

  Tasks:
  1. **A02_Part1**/A02_Part1.py
  2. **A02_Part2**/A02_Part2.py
  3. **A02_Part3**/A02_Part3.py
     Complete the function **my_main** of the Python program.
     Do not modify the name of the function nor the parameters it receives.
     The entire work must be done within Spark SQL:
     - The function my_main must start with the creation operation read above loading the dataset to Spark SQL.
     - The function my_main must finish with an action operation collect, gathering and printing by the screen the result of the Spark SQL job.
     - The function my_main must not contain any other action operation collect other than the one appearing at the very end of the function.
     - The resVAL iterator returned by collect must be printed straight away, you cannot edit it to alter its format for printing.

  4. **A02_Part4**/A02_Part4.py
     Complete the function **my_model** of the Python program.
     Do not modify the name of the function nor the parameters it receives.
     The entire work must be done within Spark Structured Streaming:
     - The function my_model must start with the creation operation readStream above loading the dataset to Spark Structured Streaming.
     - The function my_model must finish with an action operation writeStream, printing by the screen the result of the Spark Structured Streaming job.
     - The function my_model must not contain any other action operation writeStream other than the one appearing at the very end of the function.

RUBRIC.

Exercises 1-4.

- 20% of the marks => Complete attempt of the exercise (even if it does not lead to the right solution or right format due to small differences).
- 40% of the marks => Right solution and format (following the aforementioned rules) for the provided dataset.
- 40% of the marks => Right solution and format (following the aforementioned rules) for any "Additional Dataset" test case we will generate. The marks will be allocated in a per test basis (i.e., if 4 extra test are tried, each of them will represent 10% of the marks).

SUBMISSION DETAILS / SUBMISSION CODE OF CONDUCT.

Submit to Canvas by the 28st of April, 5pm.

- Submissions up to 1 week late will have 10 marks deducted.
- Submissions up to 2 weeks late will have 20 marks deducted.

On submitting the assignment you adhere to the following declaration of authorship. If you have any doubt regarding the plagiarism policy discussed at the beginning of the semester do not hesitate in contacting me.

Declaration of Authorship

I, ___ NAME___, declare that the work presented in this assignment, titled *Big Data & Analytics - Assignment 2: Spark SQL & Spark Structured Streaming*, is my own. I confirm that:

- This work was done wholly by me as part of my BSc. in Software Development or BSc. in Web Development.
- Where I have consulted the published work and source code of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this assignment source code is entirely my own work.

EXERCISE – A02 Part1 (25 marks)

Technology:

Spark SQL.

Your task is to:

- Compute the top_n_bikes with highest total duration time for their trips.

Complete the function **my_main** of the Python program.

- Do not modify the name of the function nor the parameters it receives.
- The entire work must be done within Spark SQL:
    - The function my_main must start with the creation operation 'read' above loading the dataset to Spark SQL.
    - The function my_main must finish with an action operation 'collect', gathering and printing by the screen the result of the Spark SQL job.
    - The function my_main must not contain any other action operation 'collect' other than the one appearing at the very end of the function.
    - The resVAL iterator returned by 'collect' must be printed straight away, you cannot edit it to alter its format for printing.

Results:

Output one Row per bike_id. Rows must follow decreasing order in highest total duration time for their trips. Each Row must have the following fields:

Row(bike_id, totalTime, numTrips)

EXERCISE – A02 Part2 (25 marks)

Technology:

Spark SQL.

Your task is to:

- Compute the amount of trips starting from and finishing at each station_name.

Complete the function **my_main** of the Python program.

- Do not modify the name of the function nor the parameters it receives.
- The entire work must be done within Spark SQL:
    - The function my_main must start with the creation operation 'read' above loading the dataset to Spark SQL.
    - The function my_main must finish with an action operation 'collect', gathering and printing by the screen the result of the Spark SQL job.
    - The function my_main must not contain any other action operation 'collect' other than the one appearing at the very end of the function.
    - The resVAL iterator returned by 'collect' must be printed straight away, you cannot edit it to alter its format for printing.

Results:

Output one Row per station_name. Rows must follow alphabetic order in the name of the station. Each Row must have the following fields:

Row(station, num_departure_trips, num_arrival_trips)

EXERCISE – A02 Part3 (25 marks)

Technology:

Spark SQL.

Your task is to:

- Sometimes bikes are re-organised (moved) from station A to station B to balance the amount of bikes available in both stations. A truck operates this bike re-balancing service, and the trips done by-truck are not logged into the dataset. Compute all the times a given bike_id was moved by the truck re-balancing system.

Complete the function **my_main** of the Python program.

- Do not modify the name of the function nor the parameters it receives.
- The entire work must be done within Spark SQL:
  - The function my_main must start with the creation operation 'read' above loading the dataset to Spark SQL.
  - The function my_main must finish with an action operation 'collect', gathering and printing by the screen the result of the Spark SQL job.
  - The function my_main must not contain any other action operation 'collect' other than the one appearing at the very end of the function.
  - The resVAL iterator returned by 'collect' must be printed straight away, you cannot edit it to alter its format for printing.

Results:

Output one Row per moving trip. Rows must follow temporal order. Each Row must have the following fields:

Row(start_time, start_station_name, stop_time, stop_station_name)

For example, if the dataset **contains** the following 2 trips:

- **Trip1:** A user used bike_id to start a trip from Station1 on 2019/05/10 09:00:00
  and finished the trip at Station2 on 2019/05/10 10:00:00
- **Trip2:** A user used bike_id to start a trip from Station3 on 2019/05/10 11:00:00
  and finished the trip at Station4 on 2019/05/10 12:00:00

And the dataset **does not contain** any extra trip:

- **Trip3:** A user used bike_id to start a trip from Station2 and finish at Station3
  anytime between 2019/05/10 10:00:00 and 2019/05/10 11:00:00

Then it is clear that the bike was moved from Station2 to Station3 by truck, and we output:

Row(start_time=2019/05/10 10:00:00, start_station_name=Station2, stop_time=2019/05/10 11:00:00, stop_station_name=Station3)

EXERCISE – A02 Part4. (25 marks)

Technology:

Spark Structured Streaming.

Your task is to:

- Apply *real-time* data analytics to compute the amount of trips starting from each station_name for each separate batch (file) being received. For each batch, print only the stations with, at least, min_trips departing from it.

Complete the function **my_model** of the Python program.

- Do not modify the name of the function nor the parameters it receives.
- The entire work must be done within Spark Structured Streaming:
  - The function my_model must start with the creation operation readStream above loading the dataset to Spark Structured Streaming.
  - The function my_model must finish with an action operation writeStream, printing by the screen the result of the Spark Structured Streaming job.
  - The function my_model must not contain any other action operation writeStream other than the one appearing at the very end of the function.

Results:

Each batch should print its results via a Dataframe with columns start_station_name and num_departures, using one Row per station. On batches with more than one Row, the order of the rows does not matter (i.e., the test checker **my_A02_Part4_check_results.py** will return True regardless of the order in which they are outputted).