MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

BIG DATA & ANALYTICS

ASSIGNMENT 3: OPEN BOOK EXAM

BACKGROUND.

It's the last day of 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 assignments 1 and 2 has proven the potential OptimiseYourJourney can obtain by applying MapReduce and Spark SQL to analyse large-scale public transportation datasets as the one in the New York City Bike Sharing System: https://www.citibikenyc.com/

OptimiseYourJourney

Today, you are asked to complete a new exercise (over the very same NYC dataset you have used in assignments 1 and 2) and by applying the very same techniques (sequential approach, MapReduce simulator and Spark SQL) you have used in assignments 1 and 2.

- Exercise 1: Complete the exercise using the sequential approach.
- Exercises 2 & 3: Complete the exercise using the MapReduce simulator (Exercise 2 - map stage & Exercise 3 – reduce stage).
- Exercise 4: Complete the exercise using Spark SQL.

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.

TASKS / EXERCISES.

The tasks / exercises to be completed as part of the assignment are described in the next pages of this PDF document.

- The following exercises are placed in the folder **my_code:**
  1. **A03_Part1**/A03_Part1.py
  2. **A03_Part2**/my_mapper.py
  3. **A03_Part2**/my_reducer.py
  4. **A03_Part3**/A03_Part3.py

  Marks are as follows:
  - **A03_Part1**/A03_Part1.py => 25 marks
  - **A03_Part2**/my_mapper.py => 25 marks
  - **A03_Part2**/my_reducer.py => 25 marks
  - **A03_Part3**/A03_Part3.py => 25 marks

  Tasks:
  - **A03_Part1**/A01_Part1.py
  - **A03_Part3**/A01_Part3.py
    Complete the function **my_main** of the Python program.
    Do not modify the name of the function nor the parameters it receives.

  - **A03_Part2**/my_mapper.py
    Complete the function **my_map** of the Python program.
    Do not modify the name of the function nor the parameters it receives.

  - **A03_Part2**/my_reducer.py
    Complete the function **my_reduce** of the Python program.
    Do not modify the name of the function nor the parameters it receives.

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).

TEST YOUR SOLUTIONS.

- The folder **my_results** contains the expected results for each exercise.
  - **A03_Part1**/result.txt
  - **A03_Part2/1_my_map_simulation** => 73 files for the output of each map process.
  - **A03_Part2/2_my_sort_simulation**/sort_1.txt => input for first reduce process.
  - **A03_Part2/2_my_sort_simulation**/sort_2.txt => input for second reduce process.
  - **A03_Part2/3_my_reduce_simulation**/reduce_sort_1.txt => output of first reduce.
  - **A03_Part2/3_my_reduce_simulation**/reduce_sort_2.txt => output of second reduce.
  - **A03_Part3**/result.txt

- Moreover, the subfolder **my_results/check_results** allows you to see if your code is producing the expected output or not.
  - The file **test_checker.py** needs two folders and compares if their files are equal or not.
  - When you have completed one part (e.g., A03_Part2), copy the folder **my_results/A03_Part2** into the folder **my_results/check_results/Student_Attempts/A03_Part2**.
  - Open the file **test_checker.py** and edit the line 109 with the value of the part you are attempting (e.g., part = 2).
  - Run the program **test_checker.py**. It will tell you whether your output is correct or not.

For example, as an example let's run the Python program **test_checker.py** to see if the solution attempt done by the student for A03_Part1 and A03_Part2 is correct or not.

➢ python3.9 test_checker.py 1

```
--------------------------------------------------------
Checking :
./Assignment_Solutions/A03_Part1/result.txt
./Student_Attempts/A03_Part1/result.txt

Test passed!
--------------------------------------------------------
Congratulations, the code passed all the tests!
--------------------------------------------------------
```

As we can see, the code of the student is correct, and thus it gets the marks.

➢ python3.9 test_checker.py 2

```
--------------------------------------------------------
Checking :
./Assignment_Solutions/A03_Part2/2_my_sort_simulation/sort_1.txt
./Student_Attempts/A03_Part2/2_my_sort_simulation/sort_1.txt

Test did not pass.
--------------------------------------------------------
Checking :
```

```
./Assignment_Solutions/A03_Part2/2_my_sort_simulation/sort_2.txt
./Student_Attempts/A03_Part2/2_my_sort_simulation/sort_2.txt

Test passed!
--------------------------------------------------------
Checking :
./Assignment_Solutions/A03_Part2/3_my_reduce_simulation/reduce_sort_1.txt
./Student_Attempts/A03_Part2/3_my_reduce_simulation/reduce_sort_1.txt

Test did not pass.
--------------------------------------------------------
Checking :
./Assignment_Solutions/A03_Part2/3_my_reduce_simulation/reduce_sort_2.txt
./Student_Attempts/A03_Part2/3_my_reduce_simulation/reduce_sort_2.txt

Test passed!
--------------------------------------------------------
Sorry, the output of some files is incorrect!
--------------------------------------------------------
```

As we can see, the code of the student is not correct, and thus it does not get the marks. The problem was that some output lines in some files were wrong.

Main Message

Use the program **test_checker.py** to ensure that all your exercises produce the expected output (and in the right format!).

SUBMISSION DETAILS / SUBMISSION CODE OF CONDUCT.

This open book exam is taking place on Tuesday 18th of May, 3:30pm – 6:30pm.
Submit to Canvas by the 18th of May, 6:29pm.

- Late submissions will not be accepted.

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, ___YOUR NAME___, declare that the work presented in this assignment titled 'Assignment 3: Open Book Exam' 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 and report is entirely my own work.

EXERCISE DESCRIPTION

Consider only the trips where user_type is “Subscriber”.

- Example: given the two trips below, first one is discarded and second one is considered.

start_time	**…**	**user_type**	**birth_year**	**…**
01/05/2019 08:00:00	…	Customer	1995	…
02/05/2019 09:00:00	…	Subscriber	2000	…

Consider only the trips where birth_year is bigger or equal than a parameter variable year.

- Example: if year = 1990, then given the three trips below, first one is discarded and the other two are considered.

start_time	**…**	**user_type**	**birth_year**	**…**
03/05/2019 08:00:00	…	Subscriber	1989	…
04/05/2019 09:00:00	…	Subscriber	1992	…
05/05/2019 10:00:00	…	Subscriber	1990	…

Given a trip, its start_hour is given by the two hour digits of its field start_time.

- Example: given the four trips below, their start_hour are “00”, “08”, “08” and “23”.

start_time	**…**	**user_type**	**birth_year**	**…**
06/05/2019 **00**:00:00	…	Subscriber	1990	…
07/05/2019 **08**:00:00	…	Subscriber	1991	…
08/06/2019 **08**:59:59	…	Subscriber	1990	…
09/06/2019 **23**:59:59	…	Subscriber	1995	…

We order start_hour as follows: “00” < “01” < “02” < … < “09” < “10” < … < “22” < “23”.

- Example: given three start_hour values “06”, “08” and “14” the smallest is “**06**”.

EXERCISE:

- Aggregate the trips for each different birth_year. Consider only trips where user_type is “Subscriber” and birth_year bigger or equal than a given year. For each valid birth_year compute the start_hour with most trips. Output the results by increasing birth_year.
  - Note: if a birth_year has a tie, with two or more start_hours having the very same most trips, then break the tie by selecting the smallest start_hour.

Example 1:

Given the following tiny dataset and year = 1990 then:

- Trips highlighted in **red** are to be discarded.
- Trips highlighted in **blue** are all the trips with birth_year = 1990.
- Trips highlighted in **yellow** are all the trips with birth_year = 1991.
- In this tiny dataset there are no trips for any other year.

start_time	**…**	**user_type**	**birth_year**	**…**
01/05/2019 00:00:00	…	Customer	1990	…
10/06/2019 08:30:45	…	Subscriber	1989	…
03/06/2019 **07**:30:30	…	Subscriber	1990	…
01/05/2019 **08**:00:00	…	Subscriber	1990	…
02/06/2019 **08**:59:59	…	Subscriber	1990	…
10/05/2019 **09**:00:00	…	Subscriber	1990	…
14/06/2019 **09**:25:30	…	Subscriber	1990	…
21/06/2019 **09**:59:59	…	Subscriber	1990	…
01/05/2019 **08**:00:00	…	Subscriber	1991	…
02/06/2019 **08**:30:45	…	Subscriber	1991	…
14/06/2019 **09**:25:30	…	Subscriber	1991	…

The program should output:

- birth_year 1990 ➔ start_hour = “09” ; num_trips = 3
- birth_year 1991 ➔ start_hour = “08” ; num_trips = 2

- As we can see, for the aggregation of 1990 there are:

- start_hour “07” => 1 trip ; start_hour “08” => 2 trips ; start_hour “09” => 3 trips. Therefore, the start_hour “09” is the one with most trips (3).

- As we can see, for the aggregation of 1991 there are:

- start_hour “08” => 2 trips ; start_hour “09” => 1 trip. Therefore, the start_hour “08” is the one with most trips (2).

Example 2:

The following tiny dataset is the same as for Example 1, but now it contains one extra trip highlighted with a black rectangle ▭ . Given the tiny dataset and year = 1990 then:

- Trips highlighted in red are to be discarded.
- Trips highlighted in blue are all the trips with birth_year = 1990.
- Trips highlighted in yellow are all the trips with birth_year = 1991.
- In this tiny dataset there are no trips for any other year.

start_time	**…**	**user_type**	**birth_year**	**…**
01/05/2019 00:00:00	…	Customer	1990	…
10/06/2019 08:30:45	…	Subscriber	1989	…
03/06/2019 **07**:30:30	…	Subscriber	1990	…
01/05/2019 **08**:00:00	…	Subscriber	1990	…
02/06/2019 **08**:30:25	…	Subscriber	1990	…
02/06/2019 **08**:59:59	…	Subscriber	1990	…
10/05/2019 **09**:00:00	…	Subscriber	1990	…
14/06/2019 **09**:25:30	…	Subscriber	1990	…
21/06/2019 **09**:59:59	…	Subscriber	1990	…
01/05/2019 **08**:00:00	…	Subscriber	1991	…
02/06/2019 **08**:30:45	…	Subscriber	1991	…
14/06/2019 **09**:25:30	…	Subscriber	1991	…

The program should output:

- birth_year 1990 → start_hour = "**08**" ; num_trips = **3**
- birth_year 1991 → start_hour = "08" ; num_trips = 2

- As we can see, for the aggregation of 1990 now there are:

- start_hour "07" => 1 trip ; start_hour "08" => **3** trips ; start_hour "09" => **3** trips.
  There is a tie, with start_hours "08" and "09" both with most trips (3).
  As discussed, ties are broken by selecting the smallest start_hour (in this case "08").

- As we can see, for the aggregation of 1991 the result is the same as in Example 1:

- start_hour "08" => 2 trips ; start_hour "09" => 1 trip.
  Therefore, the start_hour "08" is the one with most trips (2).

EXERCISE 1. (25 marks)

Technology:

Python (sequential approach, do not use the MapReduce simulator).

Your task is to:

- Aggregate the trips for each different birth_year. Consider only trips where user_type is "Subscriber" and birth_year bigger or equal than a given year. For each valid birth_year compute the start_hour with most trips. Output the results by increasing birth_year.
  - Note: if a birth_year has a tie, with two or more start_hours having the very same most trips, then break the tie by selecting the smallest start_hour.

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

- Do not modify the name of the function nor the parameters it receives.
- In particular, the function must read the dataset provided in input_folder and must open and write the results to output_file.
- You can use the auxiliary function process_line which, given one line from a dataset file, returns a tuple with its content.
- You can also program any other auxiliary functions you might need.

Results:

Output one text line per birth_year. Lines must follow increasing order birth_year. Each line must have the following format:

birth_year \t (start_hour, num_trips) \n

Note:

I recommend you to use a dictionary, with key → birth_year *and value → either (1) or (2):*

(1) A list of 24 items, where index → start_hour *and item value →* num_trips.
(2) A nested dictionary, with key → start_hour *and value →* num_trips.

If you want to use any other representation, it is perfectly fine.

EXERCISE 2 AND EXERCISE 3. (50 marks)

Technology:

Python - Use the MapReduce simulator.

Your task is to:

- Aggregate the trips for each different birth_year. Consider only trips where user_type is “Subscriber” and birth_year bigger or equal than a given year. For each valid birth_year compute the start_hour with most trips. Output the results by increasing birth_year.
    - Note: if a birth_year has a tie, with two or more start_hours having the very same most trips, then break the tie by selecting the smallest start_hour.

my_mapper.py => Complete the function **my_map** of the Python program.

- Do not modify the name of the function nor the parameters it receives.
- In particular, the function must read the content of a file provided by my_input_stream and must write the results to the file provided by my_output_stream. The extra parameter year is provided via my_mapper_input_parameters.
- You can use the auxiliary function process_line which, given one line from a dataset file, returns a tuple with its content.
- You can also program any other auxiliary functions you might need.

my_reducer.py => Complete the function **my_reduce** of the Python program.

- Do not modify the name of the function nor the parameters it receives.
- In particular, the function must read the content of a file provided by my_input_stream and must write the results to the file provided by my_output_stream. There are no extra parameters provided via my_reducer_input_parameters.
- You can also program any other auxiliary functions you might need.

Results:

Output one text line per birth_year. Lines must follow increasing order birth_year. Each line must have the following format:

```
birth_year \t (start_hour, num_trips) \n
```

Note:

I recommend you to use a dictionary, with key → birth_year *and value → either (1) or (2):*

1. *A list of 24 items, where index →* start_hour *and item value →* num_trips.
2. *A nested dictionary, with key →* start_hour *and value →* num_trips.

If you want to use any other representation, it is perfectly fine.

EXERCISE 4. (25 marks)

Technology:

Spark SQL.

Your task is to:

- Aggregate the trips for each different birth_year. Consider only trips where user_type is "Subscriber" and birth_year bigger or equal than a given year. For each valid birth_year compute the start_hour with most trips. Output the results by increasing birth_year.
    - Note: if a birth_year has a tie, with two or more start_hours having the very same most trips, then break the tie by selecting the smallest start_hour.

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 birth_year. Rows must follow increasing order birth_year. Each Row must have the following fields:

Row(birth_year, start_hour, num_trips)