MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

BIG DATA PROCESSING

ASSIGNMENT 1: SPARK CORE & SPARK SQL

BACKGROUND.

From the window you can see the sun shining in a lovely autumn morning. Its Monday, 10am, and you are in the open plan office of a new 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*".

The start-up is at a very early stage, and has no clear product in mind yet. However, they have offered a short-term internship in their Big Data Engineering Department to help them exploring the datasets, technologies and techniques that can be applied in their future products. They do not pay very well (0€ per hour), but you see this as a good opportunity to complement your knowledge in the module Big Data Processing you are studying at the moment, so you have decided to give it a go.

OptimiseYourJourney

In the department meeting that has just finished your boss was particularly happy. During the weekend he was exploring the public datasets provided by the Irish government at https://data.gov.ie/ and he found a transportation dataset named **Dublin Bus GPS sample data from Dublin City Council (Insight Project)**: https://data.gov.ie/dataset/dublin-bus-gps-sample-data-from-dublin-city-council-insight-project The original dataset contains 40+ millions of GPS data measurements collected from Dublin buses operating in January of 2013. Once extracted, it occupies ~5GB and is available at:
http://opendata.dublincity.ie/TrafficOpenData/sir010113-310113.zip

Your boss thinks this dataset provides a great opportunity to explore the potential of Spark Core and Spark SQL in analysing large datasets. He has already cleaned the dataset for you to perform some data analysis on it.

DATASET.

The new dataset (obtained after cleaning the original one) is provided to you in the folder **Canvas => 5_Assignments => A01_dataset.zip**.

It occupies ~3GB and contains 744 files, one per hour interval and day of the month:

- siri.201301**0100**.csv => Provides the data measurements of **01**st of January 2013 in the hour interval **[12am, 1am)**
- siri.201301**0101**.csv => Provides the data measurements of **01**st of January 2013 in the hour interval **[1am, 2am)**
- ...
- siri.201301**0108**.csv => Provides the data measurements of **01**st of January 2013 in the hour interval **[8am, 9am)**
- siri.201301**0109**.csv => Provides the data measurements of **01**st of January 2013 in the hour interval **[9am, 10am)**
- ...
- siri.201301**0123**.csv => Provides the data measurements of **01**st of January 2013 in the hour interval **[11pm, 12am)**
- siri.201301**0200**.csv => Provides the data measurements of **02**nd of January 2013 in the hour interval **[12am, 1am)**
- ...
- siri.201301**3123**.csv => Provides the data measurements of **31**st of January 2013 in the hour interval **[11pm, 12am)**

Each row of a file contains the following fields:
Date , Bus_Line , Bus_Line_Pattern , Congestion , Longitude , Latitude , Delay , Vehicle , Closer_Stop , At_Stop

- **(00) Date**
  - A String representing the date of the measurement with format <%Y-%m-%d %H:%M:%S>
  - Example: "2013-01-01 13:00:02"
- **(01) Bus_Line**
  - An Integer representing the bus line.
  - Example: 120
- **(02) Bus_Line_Pattern**
  - A String identifier for the sequence of stops scheduled in the bus line (different buses of the same bus line can follow different sequence of stops in different iterations).
  - Example: "027B1001" (it can also be empty "").
- **(03) Congestion**
  - An Integer representing whether the bus is at a traffic jam (No => 0 / Yes => 1) .
  - Example: 0
- **(04) Longitude**
  - A Float representing the longitude position of the bus.
  - Example: -6.269634
- **(05) Latitude**

    - A Float representing the latitude position of the bus.
    - Example: 53.360504
- **(06) Delay**
    - An integer representing the delay of the bus with respect to its schedule (measured in seconds). It is a negative value if the bus is ahead of schedule.
    - Example: 90.
- **(07) Vehicle**
    - An integer identifier for the bus vehicle.
    - Example: 33304.
- **(08) Closer_Stop**
    - An integer identifier for the closest bus stop.
    - Example: 7486.
- **(09) At_Stop_Stop**
    - An integer representing whether the bus vehicle is at the bus stop right now (i.e., stopping at it for passengers to hop on / hop off). (No -> 0 and Yes -> 1)
    - Example: 0.

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. A01_ex1_spark_core.py
  2. A01_ex1_spark_sql.py
  3. A01_ex2_spark_core.py
  4. A01_ex2_spark_sql.py
  5. A01_ex3_spark_core.py
  6. A01_ex3_spark_sql.py
  7. A01_ex4_spark_core.py
  8. A01_ex4_spark_sql.py
  - **Each exercise is worth 9 marks.**

    Rules:
  - On each exercise, your task is to complete the function my_main of the Python program. This function is in charge of specifying the Spark Job performing the desired data analysis.
  - When programming my_main, you can create and call as many auxiliary functions as you need.
  - Do not alter the parameters passed to the function my_main.
  - The entire work must be done "within Spark":
    - The function my_main must start with a creation operation textFile or read loading the dataset to Spark Core and Spark SQL, respectively.
    - The function my_main must finish with an action operation collect gathering and printing by the screen the result of Spark Core / 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.

- The following exercise is placed in the folder **my_code** and **my_report:**
  9. **my_code**/A01_ex5.py
     my_report/A01_report.docx
  - **The exercise and report are worth 28 marks.**
  - The report must contain a maximum of 1,000 words.

RUBRIC.

Exercises 1-8.

- 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).
- 20% of the marks => Right solution and format (following the aforementioned rules) for the "Small Dataset".

- 20% of the marks => Right solution and format (following the aforementioned rules) for the "Entire 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 2 extra test are tried, each of them will represent 20% of the marks).

Exercise 9.

- 25% of the marks => Code implementation.
- 25% of the marks => Report - Originality.
- 25% of the marks => Report - Relevance.
- 25% of the marks => Report - Technical Difficulty.

TEST YOUR SOLUTIONS.

- The folder **my_results** contains the expected results for each exercise.
- 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 files and compares whether their content is equal or not.
    - When you have completed one exercise (e.g., A01_ex1_spark_core.py), create a file with your result in the folder **my_results/check_results/Student_Solutions/**
    - Open the file **test_checker.py** and edit the lines 114 and 113 with the names of your file and the one you are comparing it against.
    - Run the program **test_checker.py**. It will tell you whether your output is correct or not.

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.

Submit to Canvas by Sunday 21st of November, 11:59pm.

- 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.

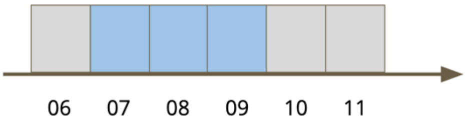
Declaration of Authorship

I, ___YOUR NAME___, declare that the work presented in this assignment titled 'Assignment 1: Spark Core and Spark SQL' is my own. I confirm that:

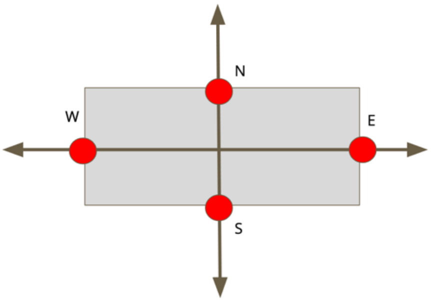
- This work was done wholly by me as part of my Msc. in Artificial Intelligence, my Msc. in Software Architecture and Design, my MSc. in Cloud Computing or my Msc. in Cyber Security at Munster Technological University.
- Where I have consulted the published work and source code of others, this is always 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 1.

Let's assume you have just got a job in Dublin. The excitement of the good news is mitigated when you start looking at prices for renting an apartment in the city centre. However, leaving aside the craziness of the house market, there is another thing you are worried about: commuting to work by bus. Traffic jams in big cities are terrible, specially by the time you go to work, on weekdays at rush hour. Your company is flexible with your starting time. The figure below highlights in blue the hour intervals your company allow you to start in the morning; in the example of the figure, any time in the hour intervals [7am, 8am), [8am, 9am) and [9am, 10am) is allowed.



To make a better decision on where to rent, you decide to draw an imaginary area around your workplace, for you to consider apartments only within that area; that is, you pick two longitude points (one in the west and one in the east) and two latitude points (one in the north and one in the south), and consider the area within these points. The figure below colours in grey the imaginary area associated to the four points selected, which are coloured in red.



You want to decide the hour interval to commute to work based on the traffic congestion of the area. The dataset you got in your hands certainly allows you to infer this information; so let's analyse it to find out.

EXERCISE: FORMAL DEFINITION

Given a program passing by parameters:

- The latitude upper bound for an area of interest, or "north" (e.g., 53.3702027)

- The longitude upper bound for an area of interest, or "east" (e.g., -6.2043634)

- The latitude lower bound for an area of interest, or "south" (e.g., 53.3343619)

- The longitude lower bound for an area of interest, or "west" (e.g., -6.2886331)

- A list of hours "hours_list" (e.g., ["07", "08", "09"])

Your task is to:

- Compute the percentage of bus measurements withing the area of interest reporting congestion. Consider only the bus measurements of weekdays (you must discard any measurement taking place on a Saturday or Sunday). Compute the results aggregating them per each hour interval.

The format of the solution computed must be:

- < (H1, P1, T1, C1), (H2, P2, T2, C2), ..., (Hn, Pn, Tn, Cn) >

where:

- Hi is a String representing the hour interval of the day (e.g, '09' - for [9am - 10am)).
- Pi is a Float representing the percentage of congestion measurements, with 2 decimal float point format (e.g., 3.54).
- Ti is an Integer representing the total number of measurements in this hour interval (e.g., 10000).
- Ci is an Integer representing the total number of measurements reporting congestion in this hour interval (e.g., 354).
- < (H1, P1, T1, C1), (H2, P2, T2, C2), ..., (Hn, Pn, Tn, Cn) > are sorted by decreasing order of Pi. Any tie in percentage is broken by sorting the entries by increasing Hi order.

EXAMPLE 1 - SMALL DATASET – A01_ex1_micro_dataset_1

Let's assume we are interested in the area limited by "north " = 53.3702027, "east " =-6.2043634, "south" = 53.3343619, "west" = -6.2886331 and "hours_list" = ["07", "08", "09"].

Let's first consider the following measurements separately to better understand the problem:

Date,Bus_Line,Bus_Line_Pattern,Congestion,Longitude,Latitude,Delay,Vehicle,Closer_Stop,At_Stop

- **2013-01-19 08:00:36**,40,015B1002,0,-6.258078,53.339279,544,33488,279,1

  This measurement is not of interest to us: weekend day.

- **2013-01-09 06:00:36**,40,015B1002,0,-6.258078,53.359279,300,33488,279,0

  This measurement is not of interest to us: hour not in hours_list.

- 2013-01-09 08:25:36,40,015B1002,0,**-6.308078**,53.359279,300,33488,279,0

  This measurement is not of interest to us: longitude is out of the area of interest.

- 2013-01-09 08:50:36,40,015B1002,0,-6.258078,**53.319279**,300,33488,279,0

  This measurement is not of interest to us: latitude is out of the area of interest.

- 2013-01-09 08:00:36,40,015B1002,**0**,-6.258078,53.339279,300,33000,279,1

  This measurement is of interest to us. The bus measurement does not report congestion.

- 2013-01-10 08:25:36,40,015B1002,**1**,-6.258078,53.339279,-200,35000,479,1

  This measurement is of interest to us. The bus measurement reports congestion.

- 2013-01-10 08:50:36,40,015B1002,**0**,-6.258078,53.339279,100,3600,279,0

  This measurement is of interest to us. The bus measurement does not report congestion.

In total we have 3 measurements for buses passing in the interval [8am - 9am): one measurement from Wednesday 9th of January 2013 and two measurements from Thursday 10th of January 2013. One of the three measurements reports congestion, so the percentage is 0.33%

The dataset A01_ex1_micro_dataset_1 contains just the following measurements:

Date,Bus_Line,Bus_Line_Pattern,Congestion,Longitude,Latitude,Delay,Vehicle,Closer_Stop,At_Stop

- 2013-01-19 08:00:36,40,015B1002,0,-6.258078,53.359279,300,33488,279,0
- 2013-01-09 06:00:36,40,015B1002,0,-6.258078,53.359279,300,33488,279,0
- 2013-01-09 08:25:36,40,015B1002,0,-6.308078,53.359279,300,33488,279,0
- 2013-01-09 08:50:36,40,015B1002,0,-6.258078,53.319279,300,33488,279,0
- 2013-01-09 08:00:36,40,015B1002,0,-6.258078,53.339279,300,33000,279,1
- 2013-01-10 08:25:36,40,015B1002,1,-6.258078,53.339279,-200,35000,479,1
- 2013-01-10 08:50:36,40,015B1002,0,-6.258078,53.339279,100,3600,279,0
- 2013-01-09 09:00:36,40,015B1002,0,-6.258078,53.339279,300,33000,279,1
- 2013-01-10 09:25:36,40,015B1002,1,-6.258078,53.339279,-200,35000,479,1
- 2013-01-10 09:50:36,40,015B1002,1,-6.258078,53.339279,100,3600,279,0

<u>SOLUTION EXAMPLE 1 - SMALL DATASET – A01_ex1_micro_dataset_1</u>

Given a program passing by parameters:

- The area’s latitude upper bound, or "north" = 53.3702027

- The area’s longitude upper bound, or "east" = -6.2043634

- The area’s latitude lower bound, or "south" = 53.3343619

- The area’s longitude lower bound, or "west" = -6.2886331

- A list of hours "hours_list" = ["07", "08", "09"]

--- SPARK CORE ---

- solutionRDD:

  < ('09', 66.67, 3, 2), ('08', 33.33, 3, 1) >

- solutionRDD printed by the screen:

  ('09', 66.67, 3, 2)

  ('08', 33.33, 3, 1)

--- SPARK SQL ---

- solutionDF:

  < Row(hour='09', percentage=66.67, numMeasurements=3, congestionMeasurements=2), Row(hour='08', percentage=33.33, numMeasurements=3, congestionMeasurements=1) >

- solutionDF printed by the screen:

```
Row(hour='09', percentage=66.67, numMeasurements=3, congestionMeasurements=2)

Row(hour='08', percentage=33.33, numMeasurements=3, congestionMeasurements=1)
```

<u>SOLUTION EXAMPLE 2 - ENTIRE DATASET – my_dataset_complete</u>

Given a program passing by parameters:

- The area's latitude upper bound, or "north" = 53.3702027

- The area's longitude upper bound, or "east" = -6.2043634

- The area's latitude lower bound, or "south" = 53.3343619

- The area's longitude lower bound, or "west" = -6.2886331

- A list of hours "hours_list" = ["07", "08", "09"]

--- SPARK CORE ---

- solutionRDD:

  < ('07', 1.1, 533547, 5852), ('08', 1.02, 758627, 7701), ('09', 0.72, 752046, 5423)>

- solutionRDD printed by the screen:

```
('07', 1.1, 533547, 5852)

('08', 1.02, 758627, 7701)

('09', 0.72, 752046, 5423)
```

--- SPARK SQL ---

- solutionDF:

  < Row(hour='07', percentage=1.1, numMeasurements=533547, congestionMeasurements=5852), Row(hour='08', percentage=1.02, numMeasurements=758627, congestionMeasurements=7701), Row(hour='09', percentage=0.72, numMeasurements=752046, congestionMeasurements=5423) >

- solutionDF printed by the screen:

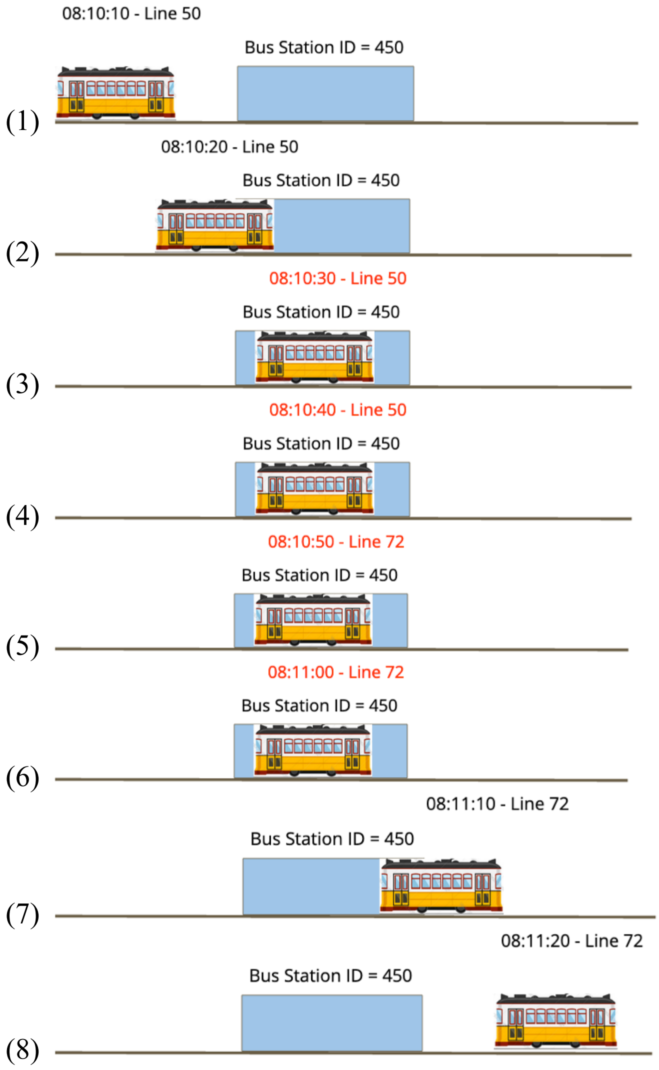
```
Row(hour='07', percentage=1.1, numMeasurements=533547, congestionMeasurements=5852)

Row(hour='08', percentage=1.02, numMeasurements=758627, congestionMeasurements=7701)

Row(hour='09', percentage=0.72, numMeasurements=752046, congestionMeasurements=5423)
```

EXERCISE 2.

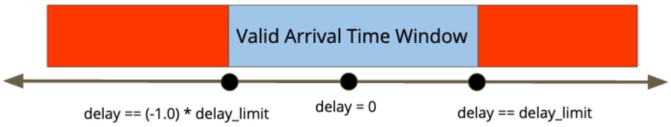
We use to think that each physical bus vehicle serves a single bus line. However, this does not need to be the case. It is perfectly possible (indeed, it happens quite often) that a concrete physical bus vehicle (e.g., "33145") serves one bus line (e.g., "25") from 8am to 2pm, and another bus line (e.g., "66") from 3pm to 9pm. A particular use-case for this could be increasing the amount of bus vehicles serving a concrete line at rush hours.

The timetable of a bus vehicle represents the sequence of bus stations it stopped at, in chronological order during the day. If consecutive chronological measurements report a bus vehicle to be stopped at a same bus station while serving a same line, then only the first of such measurements is to be considered, as it represents the arrival time of the bus vehicle for that particular station and line.

For example, in the figures below, the measurements coloured in red < (3), (4), (5), (6) > indicate the bus is actually stopped at the station. On the other hand, the measurements coloured in black < (1), (2), (7), (8) > indicate the bus is close to the station, but not stopped at it. In terms of the bus timetable, only measurements (3) and (5) are to be considered, as they represent the arrival time of the bus to station 450 while serving the lines 50 and 72, respectively. Measurements (4) and (6) can be discarded as they are continuations from (3) and (5), respectively. Finally, please note (5) is considered as it represents a transition in the line the bus vehicle is serving, from 50 to 72.



For each bus station, a fixed value or delay limit can be used to determine whether the bus vehicle arrival time is *on schedule*. For example, in the figure below the valid arrival time window is coloured in blue. If the delay value in the bus measurement associated to the bus station arrival time is within the range (-1.0) * delay_limit <= delay <= delay_limit, we say the bus is *on schedule*.



You want to compute the timetable of a concrete bus vehicle in a concrete day, and check whether it is on schedule or not for each of the stations of its timetable. The dataset you got in your hands certainly allows you to infer this information; so let's analyse it to find out.

EXERCISE: FORMAL DEFINITION

Given a program passing by parameters:

- The bus vehicle "vehicle_id" (e.g., 33145)

- The concrete day being selected "day_picked" (e.g., "2013-01-02")

- The delay threshold or "delay_limit" (e.g., 60)

Your task is to:

- Compute the timetable of vehicle_id at day_picked, including the arrival time for all bus stations it stops at during the day, in chronological order. For each of them, include the station and line being served, and use delay_limit to determine whether the bus was on schedule when arriving at the station.

The format of the solution computed must be:

- < (L1, S1, T1, O1), (L2, S2, T2, O2), ..., (Ln, Sn, Tn, On) >

where:

- Li is an Integer representing the ID of the line being served when arriving at the station (e.g., 50).
- Si is an Integer representing the ID of the station (e.g., 279).
- Ti is a String representing the arrival time at the station (e.g., "08:09:00").
- Oi in an Integer representing whether the bus was on schedule (1) or not (0) - (e.g., 0).
- The tuples (Li, Si, Ti, Oi) are sorted by increasing order of Ti.

EXAMPLE 1 - SMALL DATASET – A01_ex2_micro_dataset_1

Let's assume we are interested in "vehicle_id" = 33145, "day_picked" = "2013-01-02" and "delay_limit" = 60 (one minute, or 60 seconds).

Let's first consider the following measurements separately to better understand the problem:

Date,Bus_Line,Bus_Line_Pattern,Congestion,Longitude,Latitude,Delay,Vehicle,Closer_Stop,At_Stop

- 2013-01-02 09:00:36,40,015B1002,0,-6.258078,53.339279,544,**33245**,279,1

  This measurement is not of interest to us: wrong vehicle ID.

- **2013-01-12** 09:00:36,40,015B1002,0,-6.258078,53.339279,544,33145,279,1

  This measurement is not of interest to us: wrong day.

- 2013-01-02 08:09:00,50,015B1002,0,-6.258078,53.339279,544,33145,279,**0**

  This measurement is not of interest to us: the bus is not stopped at the station.

- **2013-01-02 08:09:00**,50,015B1002,0,-6.258078,53.339279,20,33145,279,**1**

  This measurement is of interest to us. The vehicle 33145 arrives at station 279 while serving line 50. It arrives 20 seconds late, so it is on schedule.

- **2013-01-02 08:10:00**,50,015B1002,0,-6.258178,53.339279,35,33145,450,**0**

  This measurement is not of interest to us: the bus is not stopped at the station.

- **2013-01-02 08:11:00**,50,015B1002,0,-6.258278,53.339279,-70,33145,450,**1**

  This measurement is of interest to us. The vehicle 33145 arrives at station 450 while serving line 50. It arrives 70 seconds ahead of time, so it is not on schedule.

- **2013-01-02 08:12:00**,50,015B1002,0,-6.258278,53.339279,-70,33145,450,**1**

  This measurement is not of interest to us: the vehicle continues stopped at the same station as before.

- **2013-01-02 08:13:00**,50,015B1002,0,-6.258478,53.339279,-40,33145,750,**1**

  This measurement is of interest to us: the vehicle 33145 arrives at station 750 while serving line 50. It arrives 40 seconds ahead of time, so it is on schedule.

- **2013-01-02 08:14:00**,72,015B1002,0,-6.258478,53.339279,-40,33145,750,**1**

  This measurement is of interest to us: the vehicle 33145 has changed to serve line 72, so we consider this measurement represents the vehicle to arrive at station 750 while serving line 72. It arrives 40 seconds ahead of time, so it is on schedule.

- **2013-01-02 08:15:00**,72,015B1002,0,-6.258478,53.339279,-40,33145,750,**1**

  This measurement is not of interest to us: the vehicle continues stopped at the same station as before.

- **2013-01-02 08:16:00**,72,015B1002,0,-6.258678,53.339279,100,33145,900,**1**

  This measurement is of interest to us: the vehicle 33145 arrives at station 900 while serving line 72. It arrives 100 seconds late, so it is not on schedule.

The dataset A01_ex2_micro_dataset_1 contains just the following measurements:

Date,Bus_Line,Bus_Line_Pattern,Congestion,Longitude,Latitude,Delay,Vehicle,Closer_Stop,At_Stop

- 2013-01-02 09:00:36,40,015B1002,0,-6.258078,53.339279,544,33245,279,1

- 2013-01-12 09:00:36,40,015B1002,0,-6.258078,53.339279,544,33145,279,1
- 2013-01-02 08:09:00,50,015B1002,0,-6.258078,53.339279,544,33145,279,0
- 2013-01-02 08:09:00,50,015B1002,0,-6.258078,53.339279,20,33145,279,1
- 2013-01-02 08:10:00,50,015B1002,0,-6.258178,53.339279,35,33145,450,0
- 2013-01-02 08:11:00,50,015B1002,0,-6.258278,53.339279,-70,33145,450,1
- 2013-01-02 08:12:00,50,015B1002,0,-6.258278,53.339279,-70,33145,450,1
- 2013-01-02 08:13:00,50,015B1002,0,-6.258478,53.339279,-40,33145,750,1
- 2013-01-02 08:14:00,72,015B1002,0,-6.258478,53.339279,-40,33145,750,1
- 2013-01-02 08:15:00,72,015B1002,0,-6.258478,53.339279,-40,33145,750,1
- 2013-01-02 08:16:00,72,015B1002,0,-6.258678,53.339279,100,33145,900,1

SOLUTION EXAMPLE 1 - SMALL DATASET – A01_ex2_micro_dataset_1

Given a program passing by parameters:

- The bus vehicle "vehicle_id" = 33145

- The concrete day being selected "day_picked" = "2013-01-02"

- The delay threshold or "delay_limit" = 60

--- SPARK CORE ---

- solutionRDD:

  < (50, 279, '08:09:00', 1), (50, 450, '08:11:00', 0), (50, 750, '08:13:00', 1), (72, 750, '08:14:00', 1), (72, 900, '08:16:00', 0) >

- solutionRDD printed by the screen:

  (50, 279, '08:09:00', 1)

  (50, 450, '08:11:00', 0)

  (50, 750, '08:13:00', 1)

  (72, 750, '08:14:00', 1)

  (72, 900, '08:16:00', 0)

--- SPARK SQL ---

- solutionDF:

  < Row(lineID=50, stationID=279, arrivalTime='08:09:00', onTime=1), Row(lineID=50, stationID=450, arrivalTime='08:11:00', onTime=0), Row(lineID=50, stationID=750, arrivalTime='08:13:00', onTime=1), Row(lineID=72, stationID=750, arrivalTime='08:14:00', onTime=1), Row(lineID=72, stationID=900, arrivalTime='08:16:00', onTime=0) >

- solutionDF printed by the screen:

  ```
  Row(lineID=50, stationID=279, arrivalTime='08:09:00', onTime=1)
  Row(lineID=50, stationID=450, arrivalTime='08:11:00', onTime=0)
  Row(lineID=50, stationID=750, arrivalTime='08:13:00', onTime=1)
  Row(lineID=72, stationID=750, arrivalTime='08:14:00', onTime=1)
  Row(lineID=72, stationID=900, arrivalTime='08:16:00', onTime=0)
  ```

SOLUTION EXAMPLE 2 - ENTIRE DATASET – my_dataset_complete

Given a program passing by parameters:

- The bus vehicle "vehicle_id" = 33145

- The concrete day being selected "day_picked" = "2013-01-02"

- The delay threshold or "delay_limit" = 60

--- SPARK CORE ---

- solutionRDD:

  < (171, 4747, '07:14:07', 1), (171, 6039, '07:22:23', 1), …, (38, 4745, '18:17:51', 0) >

- solutionRDD printed by the screen:

  ```
  (171, 4747, '07:14:07', 1)
  (171, 6039, '07:22:23', 1)
  …
  (38, 4745, '18:17:51', 0)
  ```

--- SPARK SQL ---

- solutionDF:

  < Row(lineID=171, stationID=4747, arrivalTime='07:14:07', onTime=1), Row(lineID=171, stationID=6039, arrivalTime='07:22:23', onTime=1), …, Row(lineID=38, stationID=4745, arrivalTime='18:17:51', onTime=0) >

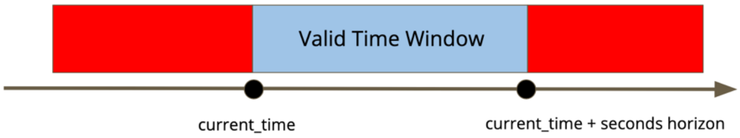
- solutionDF printed by the screen:

  ```
  Row(lineID=171, stationID=4747, arrivalTime='07:14:07', onTime=1)
  Row(lineID=171, stationID=6039, arrivalTime='07:22:23', onTime=1)
  …
  Row(lineID=38, stationID=4745, arrivalTime='18:17:51', onTime=0)
  ```

EXERCISE 3.

"Stopping advertising to save money is like stopping your watch to save time." – Henry Ford. Google and Facebook have made a fortune with their revolutionary advertisement business model, but nowadays there is still room for advertisement in the *old fashion way*. A bus station is a good place to place an ad, as it will expose it to a great amount of people, including all bus passengers hopping on/off buses there, plus any other passenger in transit and potentially seeing the ad throughout the window.

Let's suppose you want to place an ad for your company in a bus station. To reach as many people as possible, you decide to choose the bus station(s) with highest amount of bus vehicles stopping at it. A *static approach* could be to consider the entire dataset, for the ad to be permanently displayed at a bus station. Instead, you prefer a *dynamic approach*, where you focus your analysis on a small time window, for the ad to be displayed only temporarily, based on your findings. The figure below colours in blue the time window to consider, based on a given time and a seconds horizon after it.



You want to compute the bus station(s) where to place the ad for a time window. The dataset you got in your hands certainly allows you to infer this information; so let's analyse it to find out.

EXERCISE: FORMAL DEFINITION

Given a program passing by parameters:

- The start time for the time window "current_time" (e.g., "2013-01-07 06:30:00")

- The length of the time window "seconds_horizon" (e.g., 1800)

Your task is to:

- Compute the bus station(s) with highest amount of bus vehicles stopping during the time window. For a given bus station, consider only different bus vehicles (if a bus stops twice or more at the station during the time window, you only count it once).

The format of the solution computed must be:

- < (S1, L1), (S2, L2), ..., (Sn, Ln) >

where:

- Si is an Integer representing the ID of the station (e.g., 279).
- Li is an Integer list representing the bus vehicles IDs stopping at it during the time window, sorted in increasing order (e.g., [33145, 34000, 35000]).
- In case multiple stations have the very same amount of max vehicles, sort them in increasing bus station ID order.

EXAMPLE 1 - SMALL DATASET – A01_ex3_micro_dataset_1

Let's assume we are interested in "current_time" = "2013-01-07 06:30:00" and "seconds_horizon" = 1800.

Let's first consider the following measurements separately to better understand the problem:

Date,Bus_Line,Bus_Line_Pattern,Congestion,Longitude,Latitude,Delay,Vehicle,Closer_Stop,At_Stop

- **2013-01-07 06:00:00**,40,015B1002,0,-6.258078,53.339279,544,33245,279,1

  This measurement is not of interest to us: wrong time, as it does not belong to the time window.

- **2013-01-07 07:30:00**,40,015B1002,0,-6.258078,53.339279,544,33245,279,1

  This measurement is not of interest to us: wrong time, as it does not belong to the time window.

- 2013-01-07 06:45:00,40,015B1002,0,-6.258078,53.339279,544,33245,279,**0**

  This measurement is not of interest to us: the bus is not stopped at the station.

- 2013-01-07 06:46:00,40,015B1002,0,-6.258078,53.339279,544,33145,279,1

  This measurement is of interest to us: the vehicle 33145 stopped at station 279 during the time window.

- 2013-01-07 06:47:00,40,015B1002,0,-6.258078,53.339279,544,33000,279,1

  This measurement is of interest to us: the vehicle 33000 stopped at station 279 during the time window.

- 2013-01-07 06:55:00,40,015B1002,0,-6.258078,53.339279,544,33145,279,1

  This measurement is not of interest to us: the vehicle 33145 stopped at station 279 during the time window, but we have already counted it beforehand.

- 2013-01-07 06:59:00,40,015B1002,0,-6.258078,53.339279,544,30000,400,1

  This measurement is of interest to us: the vehicle 30000 stopped at station 400 during the time window.

- 2013-01-07 07:01:00,40,015B1002,0,-6.258078,53.339279,544,28000,500,1

  This measurement is of interest to us: the vehicle 28000 stopped at station 500 during the time window.

- 2013-01-07 07:03:00,40,015B1002,0,-6.258078,53.339279,544,33145,500,1

  This measurement is of interest to us: the vehicle 29000 stopped at station 500 during the time window.

  In total there are two bus stations with highest amount of bus vehicles stopping at them (2 different vehicles). There is another bus station, but it has less bus vehicles stopping at it.

The dataset A01_ex3_micro_dataset_1 contains just the following measurements:

Date,Bus_Line,Bus_Line_Pattern,Congestion,Longitude,Latitude,Delay,Vehicle,Closer_Stop,At_Stop

- 2013-01-07 06:00:00,40,015B1002,0,-6.258078,53.339279,544,33245,279,1
- 2013-01-07 07:30:00,40,015B1002,0,-6.258078,53.339279,544,33245,279,1
- 2013-01-07 06:45:00,40,015B1002,0,-6.258078,53.339279,544,33245,279,0
- 2013-01-07 06:46:00,40,015B1002,0,-6.258078,53.339279,544,33145,279,1
- 2013-01-07 06:47:00,40,015B1002,0,-6.258078,53.339279,544,33000,279,1
- 2013-01-07 06:55:00,40,015B1002,0,-6.258078,53.339279,544,33145,279,1
- 2013-01-07 06:56:00,40,015B1002,0,-6.258078,53.339279,544,30000,400,1
- 2013-01-07 06:57:00,40,015B1002,0,-6.258078,53.339279,544,28000,500,1
- 2013-01-07 06:58:00,40,015B1002,0,-6.258078,53.339279,544,33145,500,1

SOLUTION EXAMPLE 1 - SMALL DATASET – A01_ex3_micro_dataset_1

Given a program passing by parameters:

- The start time for the time window "current_time" = "2013-01-07 06:30:00"

- The length of the time window "seconds_horizon" = 1800

--- SPARK CORE ---

- solutionRDD:

  < (279, [33000, 33145]), (500, [28000, 33145]) >

- solutionRDD printed by the screen:

  (279, [33000, 33145])

  (500, [28000, 33145])

--- SPARK SQL ---

- solutionDF:

  < Row(stationID=279, sortedvehicleIDList=[33000, 33145]), Row(stationID=500, sortedvehicleIDList=[28000, 33145]) >

- solutionDF printed by the screen:

  Row(stationID=279, sortedvehicleIDList=[33000, 33145])

  Row(stationID=500, sortedvehicleIDList=[28000, 33145])

SOLUTION EXAMPLE 2 - ENTIRE DATASET – my_dataset_complete

Given a program passing by parameters:

- The start time for the time window "current_time" = "2013-01-07 06:30:00"

- The length of the time window "seconds_horizon" = 1800

--- SPARK CORE ---

- solutionRDD:

  < (794, [33435, 33470, 33484, 33490, 33556, 33559, 33587, 36019, 36022, 36065]) >

- solutionRDD printed by the screen:

  (794, [33435, 33470, 33484, 33490, 33556, 33559, 33587, 36019, 36022, 36065])

--- SPARK SQL ---

- solutionDF:

  < Row(stationID=794, sortedvehicleIDList=[33435, 33470, 33484, 33490, 33556, 33559, 33587, 36019, 36022, 36065]) >

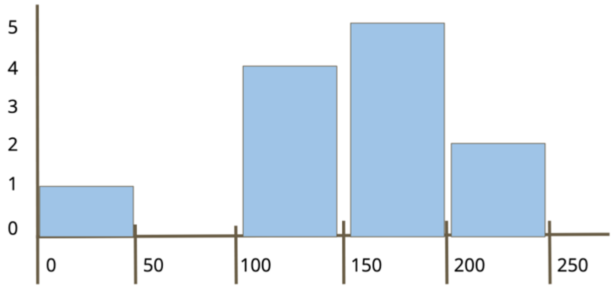
- solutionDF printed by the screen:

  Row(stationID=794, sortedvehicleIDList=[33435, 33470, 33484, 33490, 33556, 33559, 33587, 36019, 36022, 36065])

EXERCISE 4.

As with any other asset, a bus vehicle is susceptible of having technical problems precluding it from operating normally. Therefore, maintenance of the bus vehicle fleet is essential for providing a good public transport system. A very simple way of preventing a bus vehicle from breaking down is to define a *kilometre limit*, and schedule a maintenance service once the bus passes it.

You want to get an understanding of the kilometre range distribution for the bus vehicle fleet in a given day. To do so, you want to compute a histogram, where the x-axis represents *buckets* or kilometre range intervals, and the y-axis represents the amount of vehicles covering a distance within the bucket in the given day. The figure below shows the histogram of a 12 bus vehicle fleet on Monday 7th of January 2013. Buckets of 50 kilometres are sets, with 1 vehicle in the range [0, 50), 4 vehicles in the range [100, 150), 5 vehicles in the range [150, 200) and 2 vehicles in the range [200, 250).



Histogram 2013-01-07

The haversine formula is used to determine the great-circle distance between two points on a sphere, given their longitudes and latitudes. Given a day and a bus vehicle, the haversine formula is thus applied among each pair of consecutive chronological measurements. To eliminate *noise*, consecutive measurement pairs reporting an average speed over a given maximum speed are discarded.

You want to compute the histogram for the bus vehicle fleet on a given day. The dataset you got in your hands certainly allows you to infer this information; so let's analyse it to find out.

EXERCISE: FORMAL DEFINITION

Given a program passing by parameters:

- The bucket or kilometre range interval size (measured in kilometres), or "bucket_size" (e.g., = 50)

- The maximum speed accepted (measured in meters per second), or "max_speed_accepted" (e.g., 28.0)

- The day selected, or "day_picked" (e.g., "2013-01-07")

Your task is to:

- Compute the histogram for the bus vehicle fleet on a given day.

The format of the solution computed must be:

- < (I1, R1, N1), (I2, R2, N2), ..., (In, Rn, Nn) >

where:

- Ii is an Integer representing the index or bucket_ID (e.g., 0).
- Ri is a String representing the kilometre range interval for the bucket (e.g., 0-50).

- Ni is an integer representing the number of bus vehicles traversing a distance within the bucket (e.g., 5).

EXAMPLE 1 - SMALL DATASET – A01_ex4_micro_dataset_1

Let's assume we are interested in "bucket_size" = 1, "max_speed_accepted" = 28.0 and "day_picked" = "2013-01-07".

Let's first consider the following measurements separately to better understand the problem:

Date,Bus_Line,Bus_Line_Pattern,Congestion,Longitude,Latitude,Delay,Vehicle,Closer_Stop,At_Stop

- **2013-01-19 08:00:36**,40,015B1002,0,-6.258078,53.339279,544,33488,279,1

  This measurement is not of interest to us: wrong day.

- 2013-01-07 12:00:00,40,015B1002,0,-6.258000,53.359200,300,35000,300,0

  This measurement is of interest to us: let's assume this is the first measurement in the day for bus vehicle 35000. We note the location of such measurement.

- 2013-01-07 12:00:20,40,015B1002,0,-6.258800,53.359800,300,35000,350,0

  This measurement is of interest to us: assuming there is not another measurement for bus vehicle 35000 between the ones at 12:00:00 and 12:00:20, we compute the distance between the latitude and longitude of the two measurements using the Haversine formula: 85.26 meters.

  85.26 meters in 20 seconds, so the average speed in meters per second is smaller or equal to the max speed accepted. Therefore, we accumulate 85.26 meters to the distance covered by bus vehicle 35000.

- 2013-01-07 12:00:40,40,015B1002,0,-6.259800,53.351080,300,35000,400,0

  This measurement is of interest to us: assuming there is not another measurements for bus vehicle 35000 between the ones at 12:00:20 and 12:00:40, we therefore compute the distance between their latitude and longitude points using the Haversine formula: 971.89 meters.

  971.89 meters in 20 seconds, so the average speed in meters per second is bigger than the max speed accepted. Therefore, we consider these two pair of consecutive measurements as noise, and we discard the meters distance covered on them by bus vehicle 35000.

- 2013-01-07 12:01:00,40,015B1002,0,-6.259800,53.356080,300,35000,450,0

  This measurement is of interest to us: assuming there is not another measurements for bus vehicle 35000 between the ones at 12:00:40 and 12:01:00, we therefore compute the distance between their latitude and longitude points using the Haversine formula: 555.98 meters

  555.98 meters in 20 seconds, so the average speed in meters per second is smaller or equal than the max speed accepted. Therefore, we consider these two pair of consecutive measurements as noise, and we discard the 555.98 meters distance covered on them by bus vehicle 35000.

- 2013-01-07 12:01:20,40,015B1002,0,-6.259800,53.360080,300,35000,500,0

  This measurement is of interest to us: assuming there is not another measurements for bus vehicle 35000 between the ones at 12:01:00 and 12:01:20, we therefore compute the distance between their latitude and longitude points using the Haversine formula: 444.78 meters

444.78 meters in 20 seconds, so the average speed in meters per second is smaller or equal than the max speed accepted. Therefore, we consider these two pair of consecutive measurements as noise, and we discard the 444.78 meters distance covered on them by bus vehicle 35000.

In total, we have 5 valid measurements for bus vehicle 35000. When analysing them as 4 pairs of consecutive measurement, we discard one of such pairs (as the average speed was above the limit) and we sum the distance for the other 3 consecutive pairs: 85.26 + 555.98 + 444.78 = 1308.34 meters, or 1.3kms. As our bucket size is of 1km, we conclude bus vehicle 35000 belongs to the bucket [1, 2).

- 2013-01-07 12:00:00,40,015B1002,0,-6.258000,53.359200,300,36000,300,0

  This measurement is of interest to us: let's assume this is the first measurement in the day for bus vehicle 36000. We note the location of such measurement.

- 2013-01-07 12:00:20,40,015B1002,0,-6.258800,53.359800,300,36000,350,0

  This measurement is of interest to us: assuming there is not another measurement for bus vehicle 36000 between the ones at 12:00:00 and 12:00:20, we compute the distance between the latitude and longitude of the two measurements using the Haversine formula: 85.26 meters.

  85.26 meters in 20 seconds, so the average speed in meters per second is smaller or equal to the max speed accepted. Therefore, we accumulate 85.26 meters to the distance covered by bus vehicle 36000.

  In total, we have 2 valid measurements for bus vehicle 36000. When analysing it as 1 pair of consecutive measurement, we sum the distance: 85.26 meters, or 0.85kms. As our bucket size is of 1km, we conclude bus vehicle 36000 belongs to the bucket [0, 1).

The dataset A01_ex4_micro_dataset_1 contains just the following measurements:

Date,Bus_Line,Bus_Line_Pattern,Congestion,Longitude,Latitude,Delay,Vehicle,Closer_Stop,At_Stop

- 2013-01-19 08:00:36,40,015B1002,0,-6.258078,53.339279,544,33488,279,1
- 2013-01-07 12:00:00,40,015B1002,0,-6.258000,53.359200,300,35000,300,0
- 2013-01-07 12:00:20,40,015B1002,0,-6.258800,53.359800,300,35000,350,0
- 2013-01-07 12:00:40,40,015B1002,0,-6.259800,53.351080,300,35000,400,0
- 2013-01-07 12:01:00,40,015B1002,0,-6.259800,53.356080,300,35000,450,0
- 2013-01-07 12:01:20,40,015B1002,0,-6.259800,53.360080,300,35000,500,0
- 2013-01-07 12:00:00,40,015B1002,0,-6.258000,53.359200,300,36000,300,0
- 2013-01-07 12:00:20,40,015B1002,0,-6.258800,53.359800,300,36000,350,0

SOLUTION EXAMPLE 1 - SMALL DATASET – A01_ex4_micro_dataset_1

Given a program passing by parameters:

- The "bucket_size" = 1

- The "max_speed_accepted" = 28.0

- The "day_picked" = "2013-01-07"

--- SPARK CORE ---

- solutionRDD:

  < (0, '0_1', 1), (1, '1_2', 1) >

- solutionRDD printed by the screen:

  ```
  (0, '0_1', 1)
  (1, '1_2', 1)
  ```

--- SPARK SQL ---

- solutionDF:

  < Row(bucket_id=0, bucket_size='0_1', num_vehicles=1), Row(bucket_id=1, bucket_size='1_2', num_vehicles=1) >

- solutionDF printed by the screen:

  ```
  Row(bucket_id=0, bucket_size='0_1', num_vehicles=1)
  Row(bucket_id=1, bucket_size='1_2', num_vehicles=1)
  ```

SOLUTION EXAMPLE 2 - ENTIRE DATASET – my_dataset_complete

Given a program passing by parameters:

- The "bucket_size" = 50

- The "max_speed_accepted" = 28.0

- The "day_picked" = "2013-01-07"

--- SPARK CORE ---

- solutionRDD:

  < (0, '0_50', 27), (1, '50_100', 99), (2, '100_150', 137), (3, '150_200', 207), (4, '200_250', 214), (5, '250_300', 102), (6, '300_350', 31), (7, '350_400', 2), (8, '400_450', 2) >

- solutionRDD printed by the screen:

  ```
  (0, '0_50', 27)
  (1, '50_100', 99)
  ```

```
(2, '100_150', 137)

(3, '150_200', 207)

(4, '200_250', 214)

(5, '250_300', 102)

(6, '300_350', 31)

(7, '350_400', 2)

(8, '400_450', 2)
```

--- SPARK SQL ---

- solutionDF:

  ```
  < Row(bucket_id=0, bucket_size='0_50', num_vehicles=27), Row(bucket_id=1, bucket_size='50_100', num_vehicles=99), Row(bucket_id=2, bucket_size='100_150', num_vehicles=137), Row(bucket_id=3, bucket_size='150_200', num_vehicles=207), Row(bucket_id=4, bucket_size='200_250', num_vehicles=214), Row(bucket_id=5, bucket_size='250_300', num_vehicles=102), Row(bucket_id=6, bucket_size='300_350', num_vehicles=31), Row(bucket_id=7, bucket_size='350_400', num_vehicles=2), Row(bucket_id=8, bucket_size='400_450', num_vehicles=2) >
  ```

- solutionDF printed by the screen:

  ```
  Row(bucket_id=0, bucket_size='0_50', num_vehicles=27)

  Row(bucket_id=1, bucket_size='50_100', num_vehicles=99)

  Row(bucket_id=2, bucket_size='100_150', num_vehicles=137)

  Row(bucket_id=3, bucket_size='150_200', num_vehicles=207)

  Row(bucket_id=4, bucket_size='200_250', num_vehicles=214)

  Row(bucket_id=5, bucket_size='250_300', num_vehicles=102)

  Row(bucket_id=6, bucket_size='300_350', num_vehicles=31)

  Row(bucket_id=7, bucket_size='350_400', num_vehicles=2)

  Row(bucket_id=8, bucket_size='400_450', num_vehicles=2)
  ```

EXERCISE 5.

Design a novel exercise to be included in the data analysis of the Dublin Bus dataset.

Implement the novel exercise by completing the file **A01_ex5.py**. Use either Spark Core **or** Spark SQL, but not both.

Write a report of up to 1,000 words, where you discuss the novel exercise in terms of:

- Its relevance - Include a potential use-case derived from the exercise you are proposing.
- Its originality - Justify why it is different from the 4 exercises proposed.
- Its technical difficulty:
  - Briefly discuss in natural language (English and/or pseudocode) the main steps you used to implement it.
  - Justify the use of Spark Core or Spark SQL.
  - Include a small dataset to better understand the problem.