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.20130100.csv => Provides the data measurements of **01**st of January 2013 in the hour interval **[12am, 1am)**
- siri.2013010101.csv => Provides the data measurements of **01**st of January 2013 in the hour interval **[1am, 2am)**
- ...
- siri.2013010108.csv => Provides the data measurements of **01**st of January 2013 in the hour interval **[8am, 9am)**
- siri.2013010109.csv => Provides the data measurements of **01**st of January 2013 in the hour interval **[9am, 10am)**
- ...
- siri.2013010123.csv => Provides the data measurements of **01**st of January 2013 in the hour interval **[11pm, 12am)**
- siri.2013010200.csv => Provides the data measurements of **02**nd of January 2013 in the hour interval **[12am, 1am)**
- ...
- siri.2013013123.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
  - **Each exercise is worth 12.5 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_report:**
  7. A01_report.docx

  - **The report is worth 25 marks.**
  - The report must contain a maximum of 1,000 words.

RUBRIC.

Exercises 1-6.

- 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 4 extra test are tried, each of them will represent 10% of the marks).

Exercise 7.

- 25% of the marks => Originality.
- 25% of the marks => Relevance.
- 50% of the marks => Viability.

<u>TEST YOUR SOLUTIONS.</u>

- The folder **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 77 and 76 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.

<u>Main Message</u>

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

<u>SUBMISSION DETAILS / SUBMISSION CODE OF CONDUCT.</u>

Submit to Canvas by the 13th 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 Cyber Security or my Msc. in Software Architecture and Design 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 live in Dublin, and you can see from your window the bus stop where the bus bringing you to work stops at every morning. The bus passes quite regularly, and your company is flexible with your starting time; thus, you can take the bus anytime from 7am to 10am.

Traffic jams in big cities are terrible, specially by the time you go to work, on weekdays at rush hour. This affects the bus schedule, making it difficult to predict its arrival time at the bus stop. Will the bus be on schedule, or delayed/ahead of time?

Moreover, is there a concrete hour in which the bus is, on average, more reliable? For example, of the possible hours where you can take the bus < [7am - 8am), [8am - 9am), [9am – 10am) >, what is the hour with smaller average deviation w.r.t. its scheduled time?

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 stop "bus_stop" (e.g., 279)

- The bus line "bus_line" (e.g., 40)

- A list of hours "hours_list" (e.g., ["07", "08", "09"] for the intervals [7am - 8am), [8am - 9am) and [9am - 10), resp.)

Your task is to:

- Compute the average delay of "bus_line" vehicles when stopping at "bus_stop" for each hour of "hours_list" during weekdays (you must discard any measurement taking place on a Saturday or Sunday).

The format of the solution computed must be:

- < (H1, AD1), (H2, AD2), ..., (Hn, ADn) >

where:

- Hi is one of the items of "hours_list". E.g., [8am - 9am).
- ADi is the average delay for all "bus_line" vehicles stopping at "bus_stop" within that hour on a weekday.
- < (H1, AD1), (H2, AD2), ..., (Hn, ADn) > are also sorted by increasing order of ADi.

<u>EXAMPLE 1 - SMALL DATASET</u>

Let's assume we are interested in "bus_line" = 40, "bus_stop" = 279 and "hours_list" = ["08", "09"].

Let's consider 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

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

- 2013-01-09 08:00:36,**41**,015B1002,0,-6.258078,53.339279,544,33488,279,1

  This measurement is not of interest to us: Wrong bus line.

- 2013-01-09 08:00:36,40,015B1002,0,-6.258078,53.339279,544,33488,**280**,1

  This measurement is not of interest to us: Wrong bus stop.

- 2013-01-09 08:00:36,40,015B1002,0,-6.258078,53.339279,544,33488,279,**0**

  This measurement is not of interest to us: The bus is not at stop.

- **2013-01-09 18:00:36**,40,015B1002,0,-6.258078,53.339279,300,33488,279,1

  This measurement is not of interest to us: The hour is not in the list.

- 2013-01-09 08:00:36,40,015B1002,0,-6.258078,53.339279,**300**,33488,279,1

  This measurement is of interest to us. The bus reported 300 seconds delay.

- 2013-01-09 08:25:36,40,015B1002,0,-6.258078,53.339279,**-200**,33488,279,1

  This measurement is of interest to us. The bus reported -200 seconds ahead.

- 2013-01-09 08:50:36,40,015B1002,0,-6.258078,53.339279,**100**,33488,279,1

  This measurement is of interest to us. The bus reported 100 seconds delay. Now we have 3 measurements for buses passing in the interval [8am - 9am): +300 seconds, -200 seconds, +100 seconds, resp. The average delay time per bus is: + 66.67.

Let's also assume our dataset is so small that it only contains the following measurements:

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

- 2013-01-09 08:00:36,40,015B1002,0,-6.258078,53.339279,300,33488,279,1
- 2013-01-09 08:25:36,40,015B1002,0,-6.258078,53.339279,-200,33488,279,1
- 2013-01-09 08:50:36,40,015B1002,0,-6.258078,53.339279,100,33488,279,1
- 2013-01-19 08:00:36,40,015B1002,0,-6.258078,53.339279,300,33488,279,1
- 2013-01-19 08:25:36,40,015B1002,0,-6.258078,53.339279,-200,33488,279,1
- 2013-01-19 08:50:36,40,015B1002,0,-6.258078,53.339279,100,33488,279,1
- 2013-01-29 08:00:36,40,015B1002,0,-6.258078,53.339279,300,33488,279,1

- 2013-01-29 08:25:36,40,015B1002,0,-6.258078,53.339279,-200,33488,279,1
- 2013-01-29 08:50:36,40,015B1002,0,-6.258078,53.339279,100,33488,279,1
- 2013-01-09 09:00:36,40,015B1002,0,-6.258078,53.339279,100,33488,279,1
- 2013-01-09 09:25:36,40,015B1002,0,-6.258078,53.339279,-100,33488,279,1
- 2013-01-09 09:50:36,40,015B1002,0,-6.258078,53.339279,150,33488,279,1

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

--- SPARK CORE ---

- solutionRDD:

  < ('09', 50.0), ('08', 66.67) >
- solutionRDD printed by the screen:

  ('09', 50.0)

  ('08', 66.67)

--- SPARK SQL ---

- solutionDF:

  < Row(hour='09', averageDelay=50.0), Row(hour='08', averageDelay=66.67) >
- solutionDF printed by the screen:

  Row(hour='09', averageDelay=50.0)

  Row(hour='08', averageDelay=66.67)

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

Given a program passing by parameters:

- The bus stop "bus_stop" = 279

- The bus line "bus_line" = 40

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

--- SPARK CORE ---

- solutionRDD:

  < ('08', 12.09), ('07', 134.4), ('09', 331.82) >
- solutionRDD printed by the screen:

  ('08', 12.09)

  ('07', 134.4)

```
('09', 331.82)
```

--- SPARK SQL ---

- solutionDF:

  ```
  < Row(hour='08', averageDelay=12.09), Row(hour='07', averageDelay=134.4), Row(hour='09', averageDelay=331.82) >
  ```

- solutionDF printed by the screen:

  ```
  Row(hour='08', averageDelay=12.09)
  Row(hour='07', averageDelay=134.4)
  Row(hour='09', averageDelay=331.82)
  ```

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 of this could be increasing the amount of bus vehicles serving a concrete line at rush hours.

But, is this something that happens to all bus vehicles? Or, indeed something that happens every day?

Moreover, how many lines does a particular bus vehicle serve every day? Is there a particular busy day in which the bus vehicle serve a highest amount of different bus lines?

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)

Your task is to:

- Compute the day(s) of the month in which this "vehicle_id" is serving the highest amount of different bus lines, and the IDs of such bus lines.

The format of the solution computed must be:

- < (D1, LB1), (D2, LB2), ..., (Dn, LBn) >

where:

- Di is the day of the month serving maximum amount of bus lines.
- LBi is the list of bus lines served in that concrete day Di.
- LBi is sorted by increasing order in the bus line IDs.
- Note that < (D1, LB1), (D2, LB2), ..., (Dn, LBn) > will have as many pairs (Di, LBi) as days the bus vehicle is serving the very same amount of max bus lines.
- If < (D1, LB1), (D2, LB2), ..., (Dn, LBn) > contains more than one pair (Di, LBi), then the pairs are also sorted by increasing order of Di.

EXAMPLE 1 - SMALL DATASET

Let's assume we are interested in "vehicle_id" = 33145.

Let's consider the following measurements:

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

- 2013-01-19 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.

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

  This measurement is of interest to us. The vehicle 33145 serves the line 40 on Wednesday 09th of January.

- **2013-01-09 10:00:36**,40,015B1002,0,-6.258078,53.339279,544,33145,279,1

  This measurement is of interest to us, but it adds nothing new, as we already knew vehicle 33145 served the line 40 on Wednesday 09th of January.

Let's also assume our dataset is so small that it only contains the following measurements:

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

- 2013-01-04 09:00:36,25,015B1002,0,-6.258078,53.339279,544,33145,279,1
- 2013-01-08 09:00:36,122,015B1002,0,-6.258078,53.339279,544,33145,279,1
- 2013-01-08 19:00:36,120,015B1002,0,-6.258078,53.339279,544,33145,279,1
- 2013-01-09 09:00:36,66,015B1002,0,-6.258078,53.339279,544,33145,279,1
- 2013-01-09 15:00:36,25,015B1002,0,-6.258078,53.339279,544,33145,279,1
- 2013-01-09 22:00:36,67,015B1002,0,-6.258078,53.339279,544,33145,279,1
- 2013-01-17 09:00:36,39,015B1002,0,-6.258078,53.339279,544,33145,279,1
- 2013-01-17 15:00:36,67,015B1002,0,-6.258078,53.339279,544,33145,279,1
- 2013-01-17 22:00:36,66,015B1002,0,-6.258078,53.339279,544,33145,279,1
- 2013-01-28 09:00:36,67,015B1002,0,-6.258078,53.339279,544,33145,279,1
- 2013-01-28 19:00:36,25,015B1002,0,-6.258078,53.339279,544,33145,279,1
- 2013-01-30 09:00:36,122,015B1002,0,-6.258078,53.339279,544,33145,279,1

Thus, it is clear that the days our vehicle serves most lines are Wednesday 9th of January (where it serves 3 different lines, "25", "66" and "67") and Thursday 17th of January (where it also serves 3 different lines, "39", "66" and "67"). The rest of days the vehicle serves less than 3 different bus lines.

SOLUTION EXAMPLE 1 - SMALL DATASET

--- SPARK CORE ---

- solutionRDD:

  < ('09', [25, 66, 67]), ('17', [39, 66, 67]) >

- solutionRDD printed by the screen:

  ('09', [25, 66, 67])

  ('17', [39, 66, 67])

--- SPARK SQL ---

- solutionDF:

  < Row(day='09', sortedBusLineIDs=[25, 66, 67]), Row(day='17', sortedBusLineIDs=[39, 66, 67]) >

- solutionDF printed by the screen:

  Row(day='09', sortedBusLineIDs=[25, 66, 67])

  Row(day='17', sortedBusLineIDs=[39, 66, 67])

<u>**SOLUTION EXAMPLE 2 - ENTIRE DATASET**</u>

Given a program passing by parameters:

- The bus vehicle "vehicle_id" = 33145

--- SPARK CORE ---

- solutionRDD:

  < ('02', [38, 41, 171]), ('28', [13, 140, 171]), ('29', [40, 171, 238]) >

- solutionRDD printed by the screen:

  ('02', [38, 41, 171])

  ('28', [13, 140, 171])

  ('29', [40, 171, 238])

--- SPARK SQL ---

- solutionDF:

  < Row(day='02', sortedBusLineIDs=[38, 41, 171]), Row(day='28', sortedBusLineIDs=[13, 140, 171]), Row(day='29', sortedBusLineIDs=[40, 171, 238]) >

- solutionDF printed by the screen:

  Row(day='02', sortedBusLineIDs=[38, 41, 171])

  Row(day='28', sortedBusLineIDs=[13, 140, 171])

  Row(day='29', sortedBusLineIDs=[40, 171, 238])

EXERCISE 3.

When exploring a city as a tourist, there is nothing like a nice walk to observe things at a calm pace. However, in order to get a quick taste of a city, hop on and off to a bus can be the best option, especially when the distance among points of interest is very big.

Let's suppose we are in the street and we devise a bus stop. Tired as we are after a few days of tourism walking long hours, we claim to ourselves: "Ok, decision made: for the next 30 minutes today we are not going to walk anymore. What time is it now? 9am. Perfect, so until 9.30am, no walking at all! Instead, look at this bus stop across the street, let's jump in to the first bus stopping at it, and see where does this bus bring us before 9.30am".

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 current time "current_time" (e.g., "2013-01-10 08:59:59")

- A bus stop "current_stop" (e.g., 1935)

- The time you allocate to yourself before you start walking again "seconds_horizon" (e.g., 1800 seconds, which is half an hour).

Your task is to:

- Compute the first bus stopping at "current_stop" and the list of bus stops it bring us within that time "seconds_horizon".

The format of the solution computed must be:

- < (V, [(T1, S1), (T2, S2), ..., (Tn, Sn)]) >

where:

- V is the first bus vehicle stopping at "current_stop" after "current_time".
- [(T1, S1), (T2, S2), ..., (Tn, Sn)] is the list of other bus stops this bus vehicle stops at before the end of "current_time" + "seconds_horizon", with Si being a bus stop and Ti the time stopping at it.
- The list includes "current_stop" as S1, so as to know its associated value T1 (i.e., the time we hopped on the bus).

EXAMPLE 1 - SMALL DATASET

Let's assume we are interested in "current_time" = "2013-01-10 08:59:59", "current_stop" = 1935 and "seconds_horizon" = 1800.

Let's consider the following measurements:

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

- **2013-01-10 08:00:59**,40,015B1002,0,-6.258078,53.339279,544,33488,1935,1

  This measurement is not of interest to us: Its before our current time.

- **2013-01-10 10:00:59**,40,015B1002,0,-6.258078,53.339279,544,33488,1935,1

  This measurement is not of interest to us: Its after our current time + seconds_horizon.

- 2013-01-10 09:05:59,40,015B1002,0,-6.258078,53.339279,544,33488,**279**,1

  This measurement is not of interest to us: Wrong bus stop.

- 2013-01-10 09:05:59,50,015B1002,0,-6.258078,53.339279,544,33488,1935,**0**

  This measurement is not of interest to us: Bus does not stop at bus stop.

- **2013-01-20 09:05:59**,40,015B1002,0,-6.258078,53.339279,544,33500,1935,1

  This measurement is not of interest to us: Wrong day, so its after our current time + seconds_horizon.

- **2013-01-10 09:05:59**,40,015B1002,0,-6.258078,53.339279,544,33500,1935,1

  If this is the earliest bus arriving then it is interest to us: we hop on that bus.

- **2013-01-10 09:10:59**,60,015B1002,0,-6.258078,53.339279,544,33600,1935,1

  If this is not the earliest bus arriving then it is not of interest to us: we could have hopped on this bus, but another one arrived earlier, and thus we already hopped on that one.

- 2013-01-10 09:15:59,40,015B1002,0,-6.458078,53.439279,300,33500,244,**1**

  If we hopped in this vehicle, this is of interest to us: our bus is stopping at a new stop.

- 2013-01-10 09:25:59,40,015B1002,0,-6.658078,53.639279,200,33500,264,**1**

  If we hopped in this vehicle, this is of interest to us: our bus is stopping at a new stop.

- **2013-01-10 09:35:59**,40,015B1002,0,-6.858078,53.839279,100,33500,284,1

  Even if we hopped in this vehicle, this is no longer of interest to us: the time is after our current time + seconds_horizon.

Let's also assume our dataset is so small that it only contains the following measurements:

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

- 2013-01-10 08:00:59,40,015B1002,0,-6.258078,53.339279,544,33488,1935,1
- 2013-01-10 10:00:59,40,015B1002,0,-6.258078,53.339279,544,33488,1935,1

- 2013-01-10 09:05:59,40,015B1002,0,-6.258078,53.339279,544,33488,279,1
- 2013-01-10 09:05:59,50,015B1002,0,-6.258078,53.339279,544,33488,1935,0
- 2013-01-10 09:05:59,40,015B1002,0,-6.258078,53.339279,544,33500,1935,1
- 2013-01-10 09:10:59,60,015B1002,0,-6.258078,53.339279,544,33600,1935,1
- 2013-01-10 09:15:59,40,015B1002,0,-6.458078,53.439279,300,33500,244,1
- 2013-01-10 09:25:59,40,015B1002,0,-6.658078,53.639279,200,33500,264,1
- 2013-01-10 09:35:59,40,015B1002,0,-6.858078,53.839279,100,33500,284,1

As we can see:

- We hopped on bus vehicle 33500, arriving at stop 1935 at 2013-01-10 09:05:59.
- Bus vehicle 33500 arrived at stop 244 at 2013-01-10 09:15:59.
- Bus vehicle 33500 arrived at stop 264 at 2013-01-10 09:25:59.
- Bus vehicle 33500 arrived at stop 284 at 2013-01-10 09:35:59, but that was already outside of our "seconds_horizon", so we already hopped off the bus at previous stop 264.

Thus, our bus vehicle was 33500 and the stations we are interested at were:

- 2013-01-10 09:05:59, stop 1935.
- 2013-01-10 09:15:59, stop 244.
- 2013-01-10 09:25:59, stop 264.

SOLUTION EXAMPLE 1 - SMALL DATASET

--- SPARK CORE ---

- solutionRDD:

  < (33500, [('2013-01-10 09:05:59', 1935), ('2013-01-10 09:15:59', 244), ('2013-01-10 09:25:59', 264)]) >

- solutionRDD printed by the screen:

```
(33500, [('2013-01-10 09:05:59', 1935),
          ('2013-01-10 09:15:59', 244),
          ('2013-01-10 09:25:59', 264)
         ]
 )
```

Please note the RDD contains just one item, thus it is printed in just one line, with all the items of the list in a consecutive manner, not one item of the list per line. I am just formatting it here for making it more readable to you. However, once again, when you print it by the screen it should be all in one line.

--- SPARK SQL ---

- solutionDF:

  < Row(vehicleID=33500, stations=[Row(time='2013-01-10 09:05:59', stop=1935), Row(time='2013-01-10 09:15:59', stop=244), Row(time='2013-01-10 09:25:59', stop=264)]) >

- solutionDF printed by the screen:

```
Row(vehicleID=33500, stations=[Row(time='2013-01-10 09:05:59', stop=1935),
                               Row(time='2013-01-10 09:15:59', stop=244),
                               Row(time='2013-01-10 09:25:59', stop=264)
                              ]
   )
```

Please note the DF contains just one item, thus it is printed in just one line, with all the items of the list in a consecutive manner, not one item of the list per line. I am just formatting it here for making it more readable to you. However, once again, when you print it by the screen it should be all in one line.

SOLUTION EXAMPLE 2 - ENTIRE DATASET

Given a program passing by parameters:

- The current time "current_time" = "2013-01-10 08:59:59"

- A bus stop "current_stop" = 1935

- The time you allocate to yourself before you start walking again "seconds_horizon" = 1800

--- SPARK CORE ---

- solutionRDD:

  < (38022, [('2013-01-10 09:00:49', 1935), ('2013-01-10 09:04:07', 1938), ('2013-01-10 09:16:28', 1457), ('2013-01-10 09:20:47', 6094)]) >

- solutionRDD printed by the screen:

```
(38022, [('2013-01-10 09:00:49', 1935),
         ('2013-01-10 09:04:07', 1938),
         ('2013-01-10 09:16:28', 1457),
         ('2013-01-10 09:20:47', 6094)
        ]
)
```

Please note the RDD contains just one item, thus it is printed in just one line, with all the items of the list in a consecutive manner, not one item of the list per line. I am just formatting it here for making it more readable to you. However, once again, when you print it by the screen it should be all in one line.

--- SPARK SQL ---

- solutionDF:

  < Row(vehicleID=38022, stations=[Row(time='2013-01-10 09:00:49', stop=1935), Row(time='2013-01-10 09:04:07', stop=1938), Row(time='2013-01-10 09:16:28', stop=1457), Row(time='2013-01-10 09:20:47', stop=6094)]) >

- solutionDF printed by the screen (please note just one line is printed):

-
  ```
  Row(vehicleID=38022, stations=[Row(time='2013-01-10 09:00:49', stop=1935),
                                  Row(time='2013-01-10 09:04:07', stop=1938),
                                  Row(time='2013-01-10 09:16:28', stop=1457),
                                  Row(time='2013-01-10 09:20:47', stop=6094)
                                ]
     )
  ```

Please note the DF contains just one item, thus it is printed in just one line, with all the items of the list in a consecutive manner, not one item of the list per line. I am just formatting it here for making it more readable to you. However, once again, when you print it by the screen it should be all in one line.

EXERCISE 4.

Write a report of up to 1,000 words where you present and discuss:

- A novel exercise to be included in the data analysis of the Dublin Bus dataset.

There is no need to implement the new exercise, you just need to discuss it in terms of:

- Its originality - It has to be different from the 3 exercises proposed.
- Its relevance - Include a potential use-case derived from the exercise you are proposing.
- Its viability:
  - Do not implement the exercise, but briefly discuss in natural language (English and/or psudocode) the main steps that would be needed so as to implement it.
  - Include in the discussion whether, if you had to implement it, you would choose to implement it using Spark Core or using Spark SQL. Justify your selection.
  - Position the new exercise in terms of difficulty with respect to the other four exercises proposed in this assignment.