CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

COMP 9062 - Big Data
Assignment 1 - Student Submission

November 2020
Sebastian Schneider - R00183905

Assignment General Documentation:

CODE FILES

There are code files in total included in the zip file. Four for Spark Core and the same for Spark SQL. I have used Databricks for all computation. Every tasks output is also included in this submission.

Please note that regarding the SQL tasks the console outputs match as required. However the files are slightly different due to the printing elements in SQL to text (I have used the rdd.map command to do so).

NAVIGATION & STRUCTURE

For ease of use there is a "settings block" at the top of each 'may_main' function that allows to switch on the test data set and print to file. By default all are set to False.

```
####################################################
# Additional Settings:

# results dir
save_results_to_file = False
result_dir_base = "FileStore/tables/6_Assignments/"

# Utilize Test Data? (assuming here databricks only and NOT local files)
use_test_data = False
if use_test_data:
  my_dataset_dir = "/FileStore/tables/6_Assignments/ex4_small_dataset/"
  result_dir = result_dir_base + "results/ex4_small_dataset/"
else:
  result_dir = result_dir_base + "results/ex4/"

####################################################
```

Each code file is clearly commented at every step indicating what computation follows. For every major step in the SQL exercises a sample print out is also included in the code.

The remainder of this documentation is for exercise 5 only.

Exercise 5:

Novel exercise for the Dublin Bus data set.

The goal was to use some of the other data provided available that was not yet utilized in the previous exercises 1-4. The dataset offers the GPS coordinates for each reading setting up an ideal opportunity to look for localized patterns regading bus arrival delay and identify correlations.

Looking through the dataset manually I have also found some potential reading errors that could influence calculations.

For instance the highest reading of delay in seconds is 116122. Furthermore there are many more delay readings reaching in the 10k - 40k range. Those are obviously impossible and either indicate reading errors or potential continous readings of vehicles standing still. Either way those are still being being accounted for trying to reach the next stop it seems.

Furthermore I've isolated some lines manually to track their movements. I found that the indicated delay times do not always 'reset' when departing from a stop. By that I mean that the assumption would be, once a bus leaves a stop, the immediate next row should be the following stop number. However that is often not the case. Frequently when "reaching a stop" the following rows (forward in time) indicate the same stop but with the at_stop = 0 which should not be possible. I will include one findings file (findings.csv) with this submission in which I have isolated one vehicle as an example.

Either way the dataset offers a great opportunity to process the data. I have set up an own scenario that I wanted to explore and I have outlined the following three tasks:

Task A)

Find the top number of bus stops that show the highest average delay of arrival during weekdays when there is no congestion (influenced to check under 'normal' circumstances only).

Only consider records that indicate 'at stop' = 1. Take into account the given 'max_sec_delay_threshold' as the ceiling for attributed delay and discard the rest.

Regard any negative delay (early arrival) as 0, indicating an 'on time' arrival instead so as to not skew any of the negative values.

(Note: Bus stops and bus lines are represented repeatedly with the same data over and over. Cummulative results will not work but averaging (division by count) should give a relatively accurate picture instead. Hopefully.)

Task B)

What top number of bus lines are mostly affected by the delay? Use the same settings as outlined in task A.

Task C)

In relation to findings in Task A (the highest number of delayed bus stops): Cross reference the bus latitude and longitude coordinates with associated delay of bus stops.

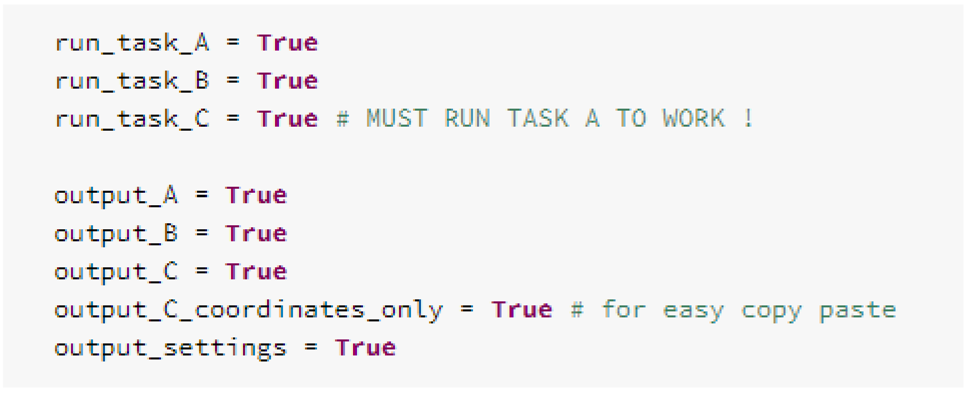
Use a pre-defined area of proximity (radius) to bundle the bus stops into groups (promity to each other). Evaluate (Google map location?) if most stops (or bus locations) with a higher delay are within the same area in the city or if the opposite is the case.

The former could indicate related traffic issues based on a few key locations only while the latter could indicate traffic issues in various locations possibly unrelated to each other (e.g.: city center vs multiple different locations).

(Note: Stops have different Lat & Long values attributed to them. Ideally use the average proximity instead as applicable.)

I am aware that an implemenation of the code was not required but I was hoping to test if the above could be done and how to best approach this scenario. The idea of cross referencing the results from Task A into actual GPS coordinates and potentially visualize is an interesting solution to a potential problem of bus delay in certain areas. This can have potential value to the bus service and public transport in general if properly investigated.

I have included the code file and will list some parts in my analysis. Please be aware that if running the file there are several flags that make task selection much easier.
The my_main function includes those:



```
run_task_A = True
run_task_B = True
run_task_C = True # MUST RUN TASK A TO WORK !

output_A = True
output_B = True
output_C = True
output_C_coordinates_only = True # for easy copy paste
output_settings = True
```

Additionally the main function (bottom of file) includes the parameters. Those can be changed as required and yield interesting results every time. I will outline one scenario here with the following parameters set:

```
if __name__ == '__main__':

    # 1. We use as many input arguments as needed
    max_sec_delay_threshold = 3600 # = 1h
    num_bus_stops = 50 # x: show top x number bus stops with highest delay (30)
    num_bus_lines = 10 # x: show top x number bus lines with highest delay (10)
    proximity_radius_km = 5 # radius in km that allows association with other Lat
```

While is seems that 1 hour delay allowance is very high it captures more severe delays that way. For a more “day to day” traffic analysis 20 minutes or less are recommended.

The number of bus stops (Task A) translates directly into Task C. The more are allowed here the longer the computation but also a increasingly better correlation can be found in Task C. The bus lines (Task B) are not related to Task C.

The promity radius sets how close GPS coordinates are allowed to be in order to be grouped as follows: In Task C all resulting GPS points are considered. All within the proximity to the next (distance will be calculated with the set radius) will be grouped and the average values (middle point) will be used for that specific group. In this way one can distinguish clearly between areas. If the radius is set to something very small (e.g.: 0.5km) it is advisable to use a higher number of bus stops to see a better correlation here.

Task A results:

Everything is printed to console. Here are the top results with the given settings in order of delay:

```
[TASK A] - The 50 highest bus stops with the most delay on average (weekdays and no congestion):

Bus stop: 816  with an average delay of: 2129 seconds (equal to 35.49 minutes or 0.59 hours).
Bus stop: 2638  with an average delay of: 2060 seconds (equal to 34.34 minutes or 0.57 hours).
Bus stop: 1955  with an average delay of: 2053 seconds (equal to 34.23 minutes or 0.57 hours).
Bus stop: 3499  with an average delay of: 1922 seconds (equal to 32.04 minutes or 0.53 hours).
Bus stop: 4204  with an average delay of: 1772 seconds (equal to 29.55 minutes or 0.49 hours).
Bus stop: 468  with an average delay of: 1513 seconds (equal to 25.23 minutes or 0.42 hours).
Bus stop: 2083  with an average delay of: 1502 seconds (equal to 25.05 minutes or 0.42 hours).
Bus stop: 5139  with an average delay of: 1369 seconds (equal to 22.83 minutes or 0.38 hours).
Bus stop: 3500  with an average delay of: 1359 seconds (equal to 22.65 minutes or 0.38 hours).
Bus stop: 815  with an average delay of: 1350 seconds (equal to 22.5 minutes or 0.38 hours).
Bus stop: 3279  with an average delay of: 1300 seconds (equal to 21.68 minutes or 0.36 hours).
Bus stop: 2052  with an average delay of: 1281 seconds (equal to 21.35 minutes or 0.36 hours).
Bus stop: 6050  with an average delay of: 1263 seconds (equal to 21.06 minutes or 0.35 hours).
Bus stop: 3749  with an average delay of: 1204 seconds (equal to 20.07 minutes or 0.33 hours).
Bus stop: 3848  with an average delay of: 1197 seconds (equal to 19.96 minutes or 0.33 hours).
Bus stop: 3660  with an average delay of: 1159 seconds (equal to 19.33 minutes or 0.32 hours).
```

Those calculated stops will be used in Task C.

Task B results:

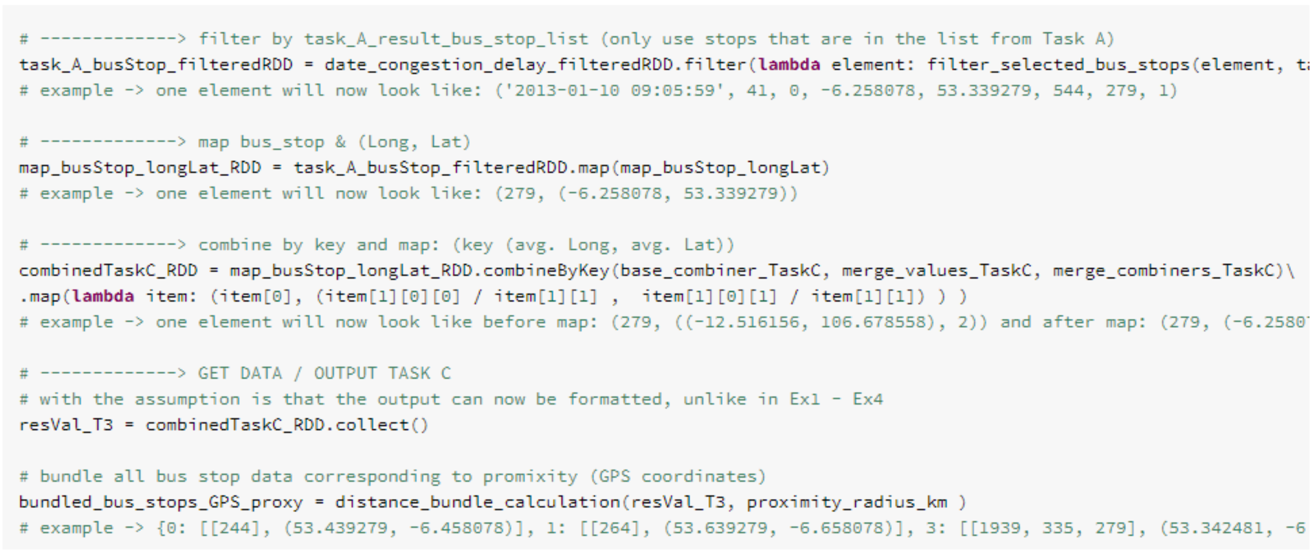
Out of interest I've decided to include the similar scenario for the bus lines overall. That way the data can be cross referenced as needed to see which stops are on which lines for example. This is the outcome for the top 10:

```
[TASK B] - The 10 highest bus lines with the most delay on average (weekdays and no congestion):

Bus line: 51  with an average delay of: 438 seconds (equal to 7.3 minutes or 0.12 hours).
Bus line: 75  with an average delay of: 413 seconds (equal to 6.89 minutes or 0.11 hours).
Bus line: 63  with an average delay of: 366 seconds (equal to 6.1 minutes or 0.1 hours).
Bus line: 116  with an average delay of: 346 seconds (equal to 5.78 minutes or 0.1 hours).
Bus line: 272  with an average delay of: 324 seconds (equal to 5.41 minutes or 0.09 hours).
Bus line: 18  with an average delay of: 321 seconds (equal to 5.36 minutes or 0.09 hours).
Bus line: 220  with an average delay of: 303 seconds (equal to 5.06 minutes or 0.08 hours).
Bus line: 67  with an average delay of: 297 seconds (equal to 4.95 minutes or 0.08 hours).
Bus line: 142  with an average delay of: 283 seconds (equal to 4.73 minutes or 0.08 hours).
Bus line: 11  with an average delay of: 272 seconds (equal to 4.54 minutes or 0.08 hours).
```

From this it becomes evident that no particular line sticks out and the delays are more less evenly split between the lines. However a more in depth analysis would be required to confirm this.

Task C results:

Implementing this Task ended up more difficult than expected as the cross referencing of GPS coordinates took some time to implement. I have used a number of auxiliary functions to achieve this. The general outline is as follows:


```
# -------------> filter by task_A_result_bus_stop_list (only use stops that are in the list from Task A)
task_A_busStop_filteredRDD = date_congestion_delay_filteredRDD.filter(lambda element: filter_selected_bus_stops(element, t
# example -> one element will now look like: ('2013-01-10 09:05:59', 41, 0, -6.258078, 53.339279, 544, 279, 1)

# -------------> map bus_stop & (Long, Lat)
map_busStop_longLat_RDD = task_A_busStop_filteredRDD.map(map_busStop_longLat)
# example -> one element will now look like: (279, (-6.258078, 53.339279))

# -------------> combine by key and map: (key (avg. Long, avg. Lat))
combinedTaskC_RDD = map_busStop_longLat_RDD.combineByKey(base_combiner_TaskC, merge_values_TaskC, merge_combiners_TaskC)\
.map(lambda item: (item[0], (item[1][0][0] / item[1][1] ,  item[1][0][1] / item[1][1]) ) )
# example -> one element will now look like before map: (279, ((-12.516156, 106.678558), 2)) and after map: (279, (-6.2580

# -------------> GET DATA / OUTPUT TASK C
# with the assumption is that the output can now be formatted, unlike in Ex1 - Ex4
resVal_T3 = combinedTaskC_RDD.collect()

# bundle all bus stop data corresponding to promixity (GPS coordinates)
bundled_bus_stops_GPS_proxy = distance_bundle_calculation(resVal_T3, proximity_radius_km )
# example -> {0: [[244], (53.439279, -6.458078)], 1: [[264], (53.639279, -6.658078)], 3: [[1939, 335, 279], (53.342481, -6
```

Applying filters and mapping is followed by combineByKey before the intial result is collected.

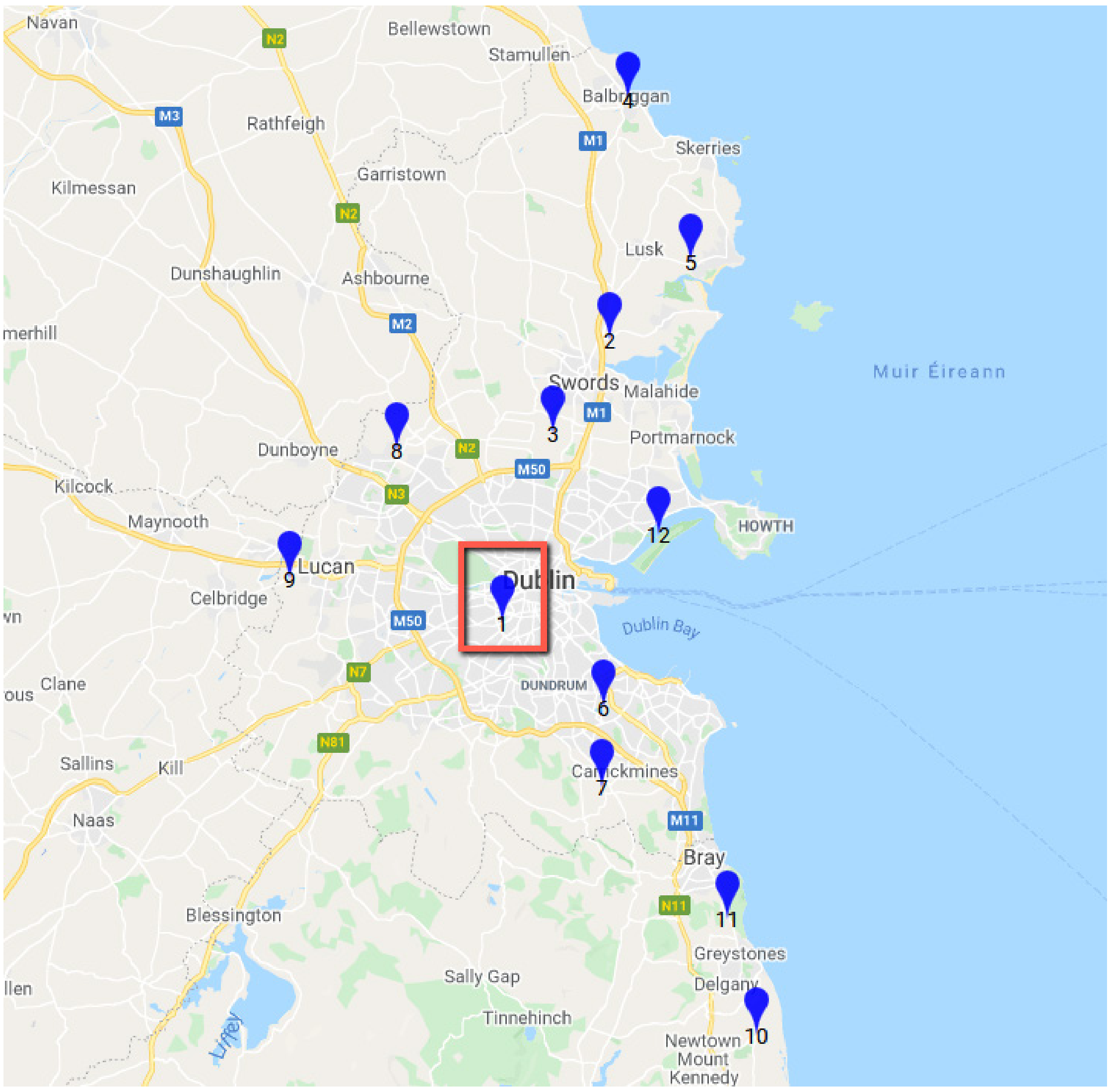
That is then crossrefenced in a very long 'distance_bundle_calculation' function (excerpt) with the results from Task A:

```python
########################### ------> WITHIN DISTANCE
# if distance is within radius -> create or append to proximity group (of bus stops)
if distance <= proximity_radius_km:

  # not empty dict -> check a few things
  if res_dict:

    # make sure bus stops aren't already in it!
    bus_stop_i_already_contained = False
    bus_stop_j_already_contained = False

    # will contain key for res_dict if found
    res_dict_i_key = 0
    res_dict_j_key = 0

    for key in res_dict: # {0: [ [bus stop(s) list], (Lat, Long)  ] }

      if resVal_T3[i][0] in res_dict[key][0]: # example: if '244' in {0: [ [264,244], ()  ] ==> True
        bus_stop_i_already_contained = True
        res_dict_i_key = key

      if resVal_T3[j][0] in res_dict[key][0]:
        bus_stop_j_already_contained = True
        res_dict_j_key = key

    # case 1: neither is yet included (add both together as a group)
    if not bus_stop_i_already_contained and not bus_stop_j_already_contained:
      # add both bus stops
      res_dict[index] = [ [ resVal_T3[i][0], resVal_T3[j][0] ] , () ]
      index += 1

    # case 2: i is included but not j
    if bus_stop_i_already_contained and not bus_stop_j_already_contained:
      # add j to i into the correct res_dict[res_dict_i_key]
      res_dict[res_dict_i_key][0].append(resVal_T3[j][0])

    # case 3: j is included but not i
    if bus_stop_j_already_contained and not bus_stop_i_already_contained:
      # add i to j into the correct res_dict[res_dict_j_key]
      res_dict[res_dict_j_key][0].append(resVal_T3[i][0])

    # case 4: j and i are included but are seperately included (CHECK!)
    # NOTE: this is a corner case that happens if both are cycled independently through an
    # "outside distance" bus stop and then added as individuals instead (happens often)
    if bus_stop_j_already_contained and bus_stop_i_already_contained:
```

The results are then bundled into regions as indicated:

```
Bus stop(s): [754, 333, 375, 1139, 2789, 2052, 2797, 2083, 4408, 2920, 1471, 2534, 2756, 1319, 2638, 1955, 7165, 2763, 2957] with (
Bus stop(s): [3749, 3748, 6050, 3856, 3859, 3860, 3862, 3863, 3660] with (avg.) location: (53.477454, -6.196067) and average delay
Bus stop(s): [816, 815] with (avg.) location: (53.427788, -6.246963) and average delay of: 1739 seconds.
Bus stop(s): [3809] with (avg.) location: (53.605904, -6.180194) and average delay of: 895 seconds.
Bus stop(s): [3845, 3843, 3846, 3848] with (avg.) location: (53.519287, -6.123873) and average delay of: 937 seconds.
Bus stop(s): [3201, 4477, 468, 4204, 3499, 3500] with (avg.) location: (53.28044, -6.201653) and average delay of: 1418 seconds.
Bus stop(s): [3282, 3279, 3521] with (avg.) location: (53.237795, -6.203456) and average delay of: 1078 seconds.
Bus stop(s): [7101] with (avg.) location: (53.418182, -6.387149) and average delay of: 911 seconds.
Bus stop(s): [7185, 3939] with (avg.) location: (53.349678, -6.483523) and average delay of: 1037 seconds.
Bus stop(s): [4265] with (avg.) location: (53.104671, -6.06499) and average delay of: 1147 seconds.
Bus stop(s): [5139] with (avg.) location: (53.167058, -6.090099) and average delay of: 1369 seconds.
Bus stop(s): [706] with (avg.) location: (53.373454, -6.152683) and average delay of: 833 seconds.
```

The highest average delay in the first region is 1098 seconds. To give a clearer picture of this plotting them to a map makes sense. The first one is naturally in the city of Dublin but many other hot spots can be identified here:



I implemented Spark Core instead of SQL. I can see the benefits of both. SQL can really help to improove the speed of calculation while I feel that I can have a better handle on the problem with the Spark Core. However this may also be due to my lesser knowledge of SQL in general.

I would be an interesting exercise to implement the same in SQL in the future and compare the computation times directly.

If more time was available I would certainly wish to experiment more with given dataset as very much can be derived from it. It is certainly a very exciting area of study.

Thank You!