Scalable and Distributed Computing

Assignment 3 - Data Analytics using PySpark

Rafaela Becerra Robalino, Marta Cortés Ocaña, Anna Subirós de Arriaga 15/03/2020

Contents

1 Introduction

The present work aims to create an efficient program that delivers some interesting information about the New York TRD, which contains records of taxi trips for January and February 2017. A serial program using Python will be done and compare against a version made with PySpark to prove the gain in terms of time analysis that it is derived by the use of this interface.

The codes for performing the described analysis can be found in detail in the files attached.

2 Theoretical Framework: Spark

Apache Spark has been cataloged as a unified analytics engine for large-scale data processing capable of handling petabytes of data. Spark will allow us to parallelize big datasets and operate on them involving less time and memory occupied. Then, as more data we have, the levels of improvement will increase.

Spark uses a master/worker architecture which involves the splitting of an application into tasks that are later distributed among the Worker nodes and put it in the waiting lines for being executed by the driver in a First-In-First-Out schedule. (Quddus, p.30 -32)

Moreover, Spark uses RDDs or Resilient Distributed Datasets which will act as objects to allow Spark to store a large amount of data into the infrastructure. The RDDs will be distributed in these multiple machines called workers but they will appear as one unified dataset, this means that parallelization and clustering will be effectuated by default. (Lai, 2019) Two types of operations can be executed on RDDs: transformations that are operations that have as a result another RDD, like map operations or wide transformations such as sorting and grouping. This kind of operations involve the data to be redistributed across partitions, in consequence, the number of operations of this type should be minimized due to its computational expensiveness. The second type of operation is referred to as actions and will return a value instead of an RDD. (Quddus, p.32)

PySpark is the Python interface for Spark which will help to create more scalable analyses in a simpler language. There are different ways to run PySpark, we will use Jupyter Nootbooks given that this tool will allow displaying the code in a friendly way.

3 Dataset and Preprocessing

The New York TRD[1] is a dataset part of the Open Data offer from the city hall which contains 1,929,670 records of taxi trips for green and yellow units provided by technology providers during the months of January and February 2017.This data set contains 17 variables which explanation can be found in Table 1.

An additionally dataset about the boroughs of NYC by their correspondence zone ID was also used to classify the pickups and drops-off locations into Boroughs. [2].

In order to obtain a cleaner, more informative and resume dataset that will allows us to extract some interesting information about the taxi trips in the city of New York, we should first make some preprocess and transformations in the variables.

Preprocessing steps:

1. Merge the two datasets.
2. Transformation of type of variables. Prioritizing the transformation of variables to categories or dates as appropriate.

[1]TLC Trip Record Data. Retrieved from: https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[2]Taxi Zone Maps and Lookup Tables. Retrieved from: https://s3.amazonaws.com/nyc-tlc/misc/taxi+$_{z}one_{l}ookup.csv$

3. Eliminate the records that present a distortion regarding the average behavior. Remove trips with zero and less than zero distance or fare payed.
4. Extract the hour of beginning of the trip based on the date time of pick up.
5. Summarize the location variables of place of pick up and drop off by introducing the classification of both by borough.
6. Create a new variable to divide trips in long and short, where a long trip will correspond to a more than 20 miles distance journey.

These steps will be performed in the serial Python programs as in the PySpark version on it, expecting at the end the same results from the calculations.

4 Methodology

4.1 Serial

After the essential first steps of importing packages and libraries (such as pandas or numpy) and loading the datasets, the next action is to apply all the functions and commands to preprocess the data in the way we have said in the previous section.

Here, we use functions as `pd.to_datetime` to convert the data to date format, `.astype` to convert the data to another type such as category, `pd.merge` to merge two datasets, `np.where` to classify a variable through a condition, among others.

In the next subsections, we talk about how to obtain or calculate the interesting information we want to know.

In particular, the indicators to be measured are:

1. The average speed for each one of the trips.
2. The average speed per hour taking as reference the pickup time.
3. The number of trips based on their trajectory taking into account its pickup and drop off locations.
4. The percentage of tip compare to the total amount of the fare paid based on the trajectory of the trip.
5. Comparison on various indicators based on the distance of the trip.

4.1.1 Average Speed per Trip and per Hour

First of all, we calculate the difference between the pickup time and the drop-off time, then, we have the time of each trip and we save it in a variable (trip_time).

We change the units of that variable to hours with the function `timedelta64` and we filter all of the values greater than zero (considering the rest as wrong records that we should discard). Finally, we divide the variable trip_distance between the variable trip_time and we obtain the speed per trip. As we have done previously, we filter all the trips discarding those which average speed is lower than 0 and higher than 150 miles per hour as we consider them are wrong records. In order to visualize the information, some plots have been done.

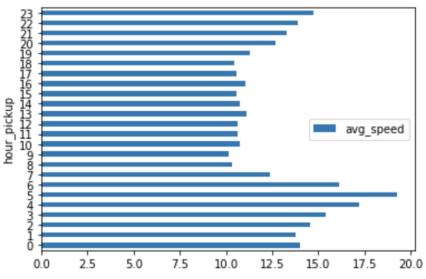

Figure 1: Average speed per hour of pickup

Figure 1, shows how the speed of the trips is distributed in the NYC taxi fleet.

Regarding the average speed per hour, the operation to be done is quite simple, we only apply a `group_by` function regarding the mean of each hour of the day taking as reference the pickup time.

In the next graph we can see that the hour when the taxis have to drive slowest is at 9 am (surely, because the early-morning traffic) and the hour when taxis go faster is at 5 am and generally during the night hours.The complete results can be found in the appendix, table 2.

4.1.2 Number of trips per location and percentage of tips

Here, we also apply a `group_by` function over the pickup and drop-off borough and count the records with the function `size.reset_index`. We create, then, a new column called 'count'. The trip with less frequency is between EWK and Manhattan and the trip with more frequency is inside Manhattan. Results about this point can be found in the appendix, table 3.

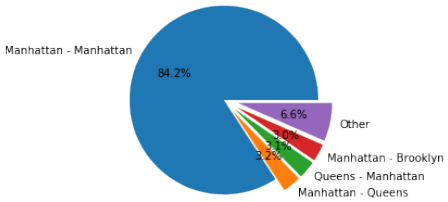

Figure 2: Four most demanded trajectories

For calculating the percentage of tips we apply `group_by` per borough regarding the sum with the function `sum.reset_index` and we consider the tip amount and the total fare of the trip dividing the first between the second and sorting it in descending way.

The next graph shows a bar plot of the percentage of tip of the total trips compared with the number of trips of each of the trajectory.

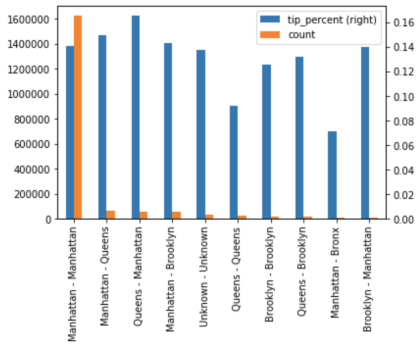

Figure 3: Number of trips and percentage of tips by trajectory

4.1.3 Indicators based on the distance of the trip

Lastly, in order to compute some comparison about features of long and short trips, we have grouped the data by the mean of all the variables. (Refer to Table 4 of the appendix)

Then, some of the insights that we can identify are:

- The mean of passengers is higher for long trips (1.71) than for short trips (1.64), although both values are very similar.
- Short trips are payed in cash more often than long trips which are payed more by credit card.
- Obviously, the mean fare amount of long trips is immensely higher than the one for the short trips, specifically, around 4 times more. On the contrary, the mean of tips in long trips is also much bigger, however, in comparison, it is lower in percentage than the mean for the short trips (13,6% and 14,2%, respectively).

4.2 PySpark

The results obtained in the serial part are the same that the ones obtained in the Pyspark part. Hence, in this section we will only explain the methodology of how to obtain those results. In the PySpark code, first of all, we need to start a PySpark Session and we will need to use `findspark` library and finally start the session. With this done, we can import the two datasets from the CSV as PySpark data frames and also the zones dataset used in the serial part. The same preprocessing that is done in a serial program is also done in the PySpark part with some functions that can be found in the `pyspark.sql.functions` library:

- **to_timestamp**: Used to convert the dataframe column type
- **col**: Easy way to access a column by name using PySpark dataframes
- **hour**: Obtain the hour from a datetime column
- **unix_timestamp**: Convert a timestamp in unix format to realize operations with timestamp like subtract.
- **when**: Used to create a new column based on a condition of other columns.

Additionally, some other base PySpark functions are used such as union, filter, withColumn or join in order to get the same preprocessing that has been done with PySpark.

4.2.1 Average Speed per Trip and per Hour

To analyze the average speed per trip, we will not use map-reduce as it involves the creation of a new column in the dataset that will be used in posterior analysis. To do this, we will generate a new column filled with the division of the trip distance and the trip duration.

However, to calculate the average speed per hour, we will use Map-Reduce. The first step is to convert the dataframe to RDD with the following structure: the key is the pickup hour and the value is the average speed for each of the rides. Once we have this, we do a map where we create a tuple of the average speed and a 1 in order to do a posterior reduce. In this case mapValues is used instead of map as only the values of the RDD are modified. The following step is to do the reduce by key which will be done by adding the two values from the previous tuple. Finally, another mapValues is done to get the average by dividing both values of the tuple.

4.2.2 Number of trips per location and percentage of tips

For this analysis, we will also proceed to do a map reduce as it has been done previously. As we are calculating the number of trips between zones and the percentage of tips over the fare amount also grouped by the zone, we are going to use the pickup zone and dropoff zone as the key, in a tuple. Regarding to the values, we are mapping three values for the same key: the tip amount and the fare amount (in order to get the percentage) and the number 1 to make possible the count of the number of trips. Then, a reduceByKey is done by adding the three values. Finally, we do a final map of the values to calculate the

average by dividing the sum of the tip amount by the sum of the fare amount by pickup-dropoff location. As expected, the results of the analysis are the same than in the serial case.

4.2.3 Indicators based on the distance of the trip

Following the previous procedure, we are transforming the PySpark dataframe to RDD and we map as follows: the key is the column created regarding the short or long type of the trip and the values are the following: The count of passengers, the fare amount, the tip amount, the tolls amount, the total amount and the average speed. However, we add a one in order to generate a count of how many trips are of each of the types in order to calculate the average. The next step is the reduction by key where we add all the values that have been previously commented. Finally, we map the values to obtain the average by different values over the total number of trips in the two types.

It should be noted that the graphics embedded in the serial part of this report have been prepared in the same way for the serial program to create two different approaches for getting the same results.

5 Results

The speed of processing the data will be derived by the amount of data process and the time spend by the programs. In both cases the amount of data will be the equivalent to the combined size of the files: Trips January 2017 (83,552 KB), Trips February 2017 (83,539 KB, and the zone data (11 KB), hence the total is 167,102 KB or 167,10 MB.

In Table 1, it can be observed the execution time that has been obtained for each part of the code, comparing the performance, individually and in total, of the serial version versus the PySpark one.

In order to make the same comparison of the data we are not taking into account the printing of the results in the Jupyter Notebook and we are separating the code in: preprocessing part, the creation of the indicators, and the transformations needed for the visualization and the plot generation. Because, as said before, in the case of PySpark, to plot the result, the RDD has to be transformed to a Panda Data Frame and finally the plot will be generated with the code that has been used in the serial.

As seen, the serial program performs better in the preprocess part and when doing the transformations and plots. On the other hand, Pyspark can be much quicker for tasks that are performed with RDDs and can be paralellized. If we measure the total amount of time, the speed for the whole program will be 16 MB/sec, while the PySpark version reaches a lower speed of 3.06 MB/sec. Nevertheless, if we do not take into account the last part, we will see that the Pyspark version is faster than the serial part.

Part of the Process	**Serial**	**Pyspark**
Preprocessing	9.2374	12.9057
Average speed by pick up hour	1.4306	0.9677
Number of trips and average of tips by zone	0.6931	0.0284
Type of travel analysis	0.3270	0.0441
Transformation and plot	0.4058	39.8678
TOTAL	10.4389	54.5876

Table 1: Summary of time results (in seconds) of the different parts of the code and in total

6 Conclusions

Due to the fact that Spark supports parallel distributed processing of data, and uses RDD which can be seen as logical partitions that are computed on different nodes, the computational time spend of

performing the same task can be highly reduced. This, in fact, can be detected by reviewing the time spend on getting the information for the analysis.

Conversely, the preprocessing part and the transformations and plot took longer in the PySpark program. This is explained by the fact that transformations were processed in the local session of PySpark. Moreover, plotting libraries are not expected to run in multiple machines, then the RDD needs to be brought to the local session as well.

Consequently, the advantages of PySpark can be evident when the aim is to process big data in tasks that can be parallelized, performing better in terms of time consumption than simpler Python Language.

Regarding the average speed per hour based on the pickup hour, we can see that at wee hours at night and in the early morning the average speed increases, being able to identify a high peak at 5 am.

Based on the borough of the pickup location and the drop off location it can be said that most of the journeys are effectuated in the area of Manhattan. Additionally, there are other three trajectories that are popular: Manhattan-Queens, Queens-Manhattan, and Manhattan-Brooklyn.

If we look at the percentage of tip that customers gave in each trajectory, we can see that the trajectory from Bronx locations to EWR Airport ended with the highest percentage of tip over the total fare paid, which duplicated it buy been 100%, nevertheless, these were only two trips. Other high percentages can be seen to have this same behavior, but there are some other trajectories that presented high rates, between 15% and 16%, and that was formed by a large number of trips. In this group, we find the route Queens-Manhattan with 59,000 trips and 16.5% of tips, Manhattan-EWR airport with 14.98%, and Manhattan-Queens with 61,378 trips and 14.97%. An important highlight is that the journeys in the area of Manhattan which are most of the trips as said were not the ones with a higher tip rate and presented just 14% of total tips from the amount of fare paid.

By classifying the trips in short and large distances, we can identify some interesting patterns. For example, if we measure the mean number of passengers we can see that it does not vary among these groups, but there are other indicators that change like the tolls paid, average speed, the total fare and, consequently, the total amount paid.

7 Bibliography

[1] Apache Spark. (2020). Retrieved from: https://spark.apache.org/

[2] Spark Python API Docs. (2020). Retrieved from: https://spark.apache.org/docs/latest/api/python/index.html

[3] Lai, R. (2019). Hands-On Big Data Analytics with Pyspark: Analyze Large Datasets and Discover Techniques for Testing, Immunizing, and Parallelizing Spark Jobs.

[4] Quddus, J. (n.d.). Machine learning with Apache Spark quick start guide: Uncover patterns, derive actionable insights, and learn from big data using MLlib. Packt Publishing.

8 Appendix

8.1 Table 1: Description of variables

Variable name	Description
VendorID	Provider of the record.
tpep_pickup_datetime	The date and time of the trip pick-up
tpep_dropoff_datetime	The date and time of the trip drop-off
passenger_count	The number of passengers in the vehicle
trip_distance	The elapsed trip distance in miles reported by the taximeter.
RatecodeID	The final rate code in effect at the end of the trip. 1 = Standard rate; 2 = JFK; 3 = Newark; 4 = Nassau or Westchester; 5 = Negotiated fare; 6 = Group ride.
store_and_fwd_flag	Indicates if the trip was a part of a shared ride. Y = store and forward trip; N = not a store and forward trip.
PULocationID	TLC Taxi Zone in which the trip began.
DOLocationID	TLC Taxi Zone in which the trip ended.
payment_type	A numeric code signifying how the passenger paid for the trip. 1 = Credit card; 2 = Cash; 3 = No charge; 4 = Dispute; 5 = Unknown; 6 = Voided trip.
fare_amount	The time-and-distance fare calculated by the meter.
extra	$0.50and 1 rush hour and overnight charges.
mta_tax	$0.50 MTA tax.
tip_amount	Tips that were not in cash.
tolls_amount	Total amount of all tolls paid in trip.
improvement_surcharge	$0.30 improvement surcharge.
total_amount	The total amount charged.

8.2 Table 2: Average Speed per Hour

Hour of pickup	Average Speed (miles/hour)
0	14.026812
1	13.780904
2	14.561261
3	15.428429
4	17.222582
5	19.248043
6	16.10032
7	12.390304
8	10.303333

9	10.126495
10	10.724652
11	10.611784
12	10.659101
13	11.091529
14	10.731254
15	10.58766
16	11.055887
17	10.54632
18	10.424866
19	11.299768
20	12.663602
21	13.303486
22	13.920524
23	14.749824

8.3 Table 3: Number of trips per trajectory and percentage of tips

Trajectory	Number of trips	Percentage of tip
Bronx - EWR	2	100.66%
Staten Island - Manhattan	1	30.65%
EWR - Manhattan	1	23.26%
EWR - EWR	25	22.46%
EWR - Brooklyn	1	21.58%
Brooklyn - Unknown	49	19.37%
Queens - Manhattan	59,001	16.55%
Manhattan - EWR	3,088	16.26%
Manhattan - Queens	61,378	14.99%
Unknown - Brooklyn	149	14.98%
Brooklyn - EWR	21	14.54%
Manhattan - Brooklyn	58,047	14.34%
Manhattan - Staten Island	280	14.21%
Manhattan - Manhattan	1,623,207	14.09%
Brooklyn - Manhattan	8,604	14.02%
Unknown - Manhattan	3,263	13.97%
Unknown - Unknown	29,207	13.78%
Queens - Brooklyn	14,630	13.18%
Manhattan - Unknown	2,364	12.95%
Unknown - EWR	25	12.92%
Brooklyn - Brooklyn	20,064	12.53%
Brooklyn - Queens	1,989	12.13%
Unknown - Queens	256	12.05%
Queens - Unknown	1,508	12.05%
Bronx - Unknown	27	10.46%
Staten Island - Staten Island	16	9.21%
Queens - Queens	27,501	9.20%
Queens - EWR	55	9.15%
Brooklyn - Staten Island	13	8.64%
Bronx - Brooklyn	23	7.77%

Unknown - Bronx	34	7.69%
Queens - Bronx	1,637	7.38%
Manhattan - Bronx	9,616	7.12%
Bronx - Queens	51	7.05%
Queens - Staten Island	101	6.69%
Brooklyn - Bronx	67	6.10%
Bronx - Manhattan	601	5.25%
Bronx - Bronx	1,225	4.07%
EWR - Unknown	1	0.00%

8.4 Table 4: Indicators based on the distance of the trip

Type of journey	Mean passenger count	Mean distance	Payment Type	Mean fare amount ($)	Mean tip amount ($)	Mean toll amount ($)	Mean total amount ($)	Average speed
Long	1.71	22.99	1.30	61.40	8.37	4.80	75.75	31.22
Short	1.64	2.76	1.33	12.10	1.72	0.25	15.17	11.91