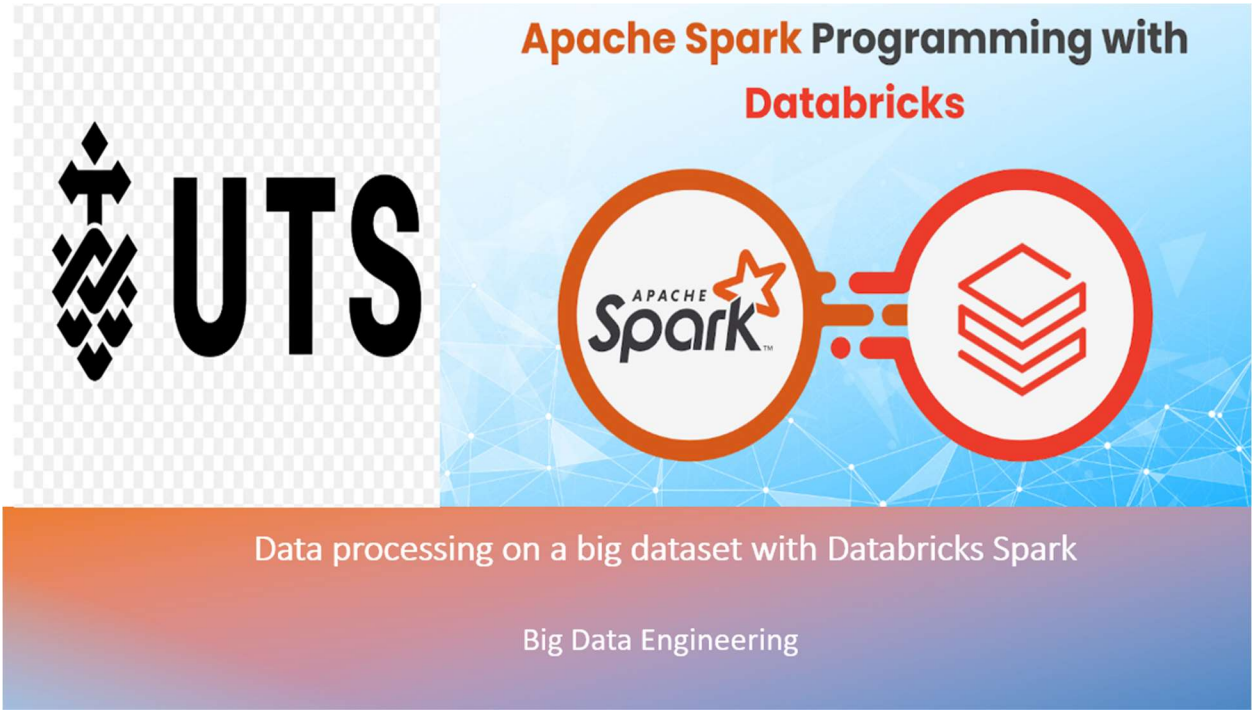
Big Data Engineering

Assignment 2: Data processing on a big dataset with Databricks Spark



The New York City Taxi and Limousine Commission (TLC) is the agency responsible for licencing and regulating New York City's taxi cabs since 1971.

TLC records millions of trips for both yellow and green taxi cabs, and these records have details about pick-up and drop-off dates, times, locations of pick-up and drop, trip distances, itemised fares, rate types, payment types, and driver-reported passenger counts. TLC has published all the records holding these details.

Introduction to the dataset

The New York City Taxi and Limousine Commission (TLC) is the agency responsible for licencing and regulating New York City's taxi cabs since 1971. TLC has publicly published millions of trip records from both yellow and green taxi cabs.

Each record includes fields capturing pick-up and drop-off dates and times, locations, trip distances, itemised fares, rate types, payment types, and driver-reported passenger counts.

Yellow taxi cabs are the iconic taxi vehicles from New York City that have the right to pick up street-hailing passengers anywhere in the city. There are around 13,600 authorised taxis in New York City, and each taxi must have a yellow medallion affixed to it.

Green taxis were introduced in August 2013 to improve taxi service and availability in the boroughs. Green taxis may respond to street hail, but only in certain designated areas.

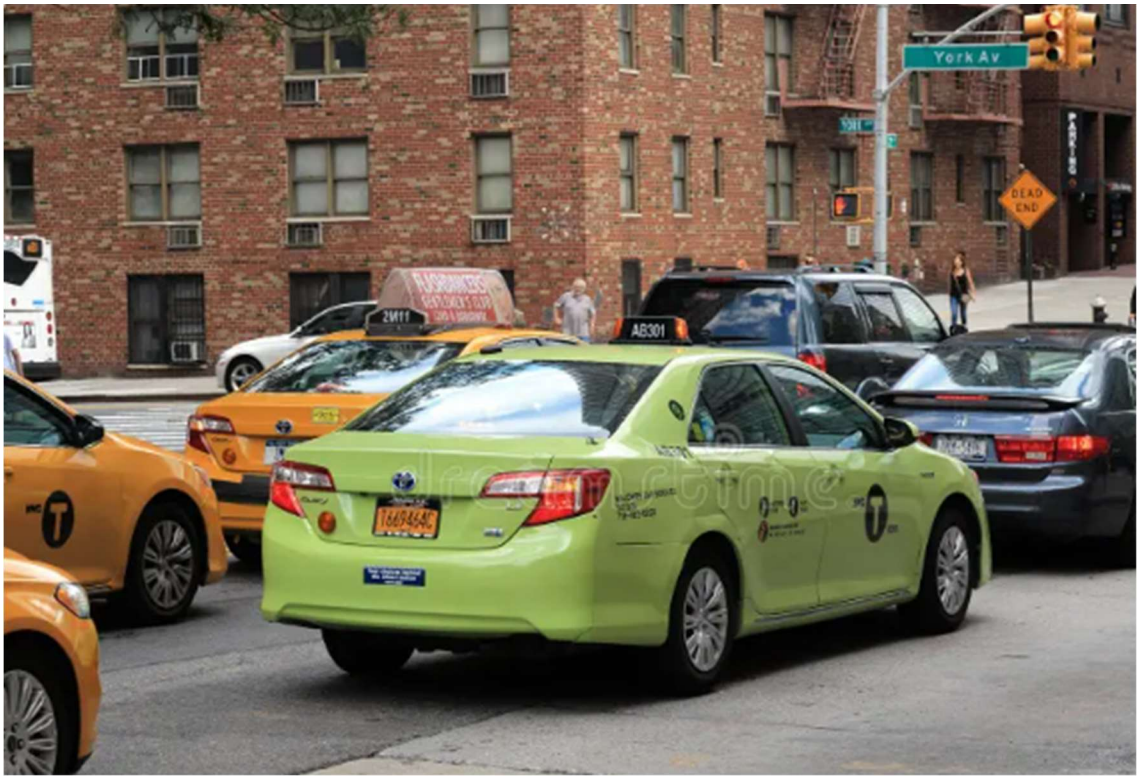


Image: Yellow and Green taxi

PART 1: Data Ingestion and Preparation

Microsoft Azure: It is a cloud computing platform run by Microsoft.
The Azure storage is created to store the parquet, csv file.
The container is used to load files into it in order to access these files from an external source.

Databricks: It is a unified, open analytics platform for building, deploying, sharing and maintaining enterprise-grade data, analytics, and AI solutions at scale. The Azure Databricks Lakehouse Platform integrates with cloud storage and provides security to the cloud account, it also manages and deploys cloud infrastructure.

Apache Spark: Apache Spark is a engine that supports large data processing. It is a open source framework that is run on various storage systems such as Amazon S3, HDFS, Cassandra, and others. Spark support memory caching, interactive queries, machine learning models, ETL process etcs. Spark has libraries for python, SQL, machine learning, graph computing, and stream processing that can be used together in an application.

Integration process of Azure storage with Databricks:

- Spark session is initialized to start the session.
- Each time the spark session is initiated, cluster is assigned to the session.
- The size of each cluster is 15.25 gb, and this cluster can run freely upto 15.25gb data.
- The spark version used in the project is 3.3.2.
- To build connection between Azure storage and databricks and to perform queries using Apachi spark, following information is needed:
- storage_account_name = "utsbde"
- blob_container_name = "assignment-2"
- storage_account_access_key = "S+MWogzrFCTaK8rF608ldaykMQ9+ckHyREvbH6sXMxUkTAmM+”
- Then azure storage should be mounted to access the data from azure using the code:

```
dbutils.fs.mount(source=f'wasbs://{blob_container_name}@{storage_account_name}.blob.core.windows.net, mount_point = f'/mnt/{blob_container_name}/',extra_configs = {'fs.azure.account.key.' + storage_account_name + '.blob.core.windows.net': storage_account_access_key})
```

- To break the connection from azure temporary drive, it should be unmounted using code dbutils.fs.unmount("/mnt/assignment-2").
- dbutils.fs.ls to list the files in the blob.
- The catch function is used to store the data temporarily so that it can be easily retrieved when the code is re-run.
- The write function is used to write the parquet and csv files into the Databricks DBFS, while the read function is used to read the files stored in the DBFS.
- The function "count" helps in finding the number of rows in the dataframe, while the "len.dataframe.columns" function gives the number of columns present in the dataframe. There are 66,200,401 rows and 20 columns in the green taxi parquet file,

```
The number of rows in green taxi parquit file are: 19233765
The number of columns present in the green taxi parquet: 20
```

Figure :Number of rows and columns in green taxi data

Table +

	VendorID	lpep_pickup_datetime	lpep_dropoff_datetime	store_and_fwd_flag	RatecodeID	PULocation
1	2	2015-05-01T00:24:18.000+0000	2015-05-01T00:24:39.000+0000	N	1	146
2	2	2015-05-01T00:28:15.000+0000	2015-05-01T00:29:00.000+0000	N	1	146
3	2	2015-05-01T00:07:45.000+0000	2015-05-01T00:12:50.000+0000	N	1	255
4	2	2015-05-01T00:26:23.000+0000	2015-05-01T00:55:12.000+0000	N	1	255
5	2	2015-05-01T00:20:16.000+0000	2015-05-01T00:30:37.000+0000	N	1	80
6	2	2015-05-01T00:37:21.000+0000	2015-05-01T00:50:00.000+0000	N	1	37
7	2	2015-05-01T00:06:15.000+0000	2015-05-01T00:31:20.000+0000	N	1	255

10,000 rows | Truncated data | 41.63 seconds runtime Refreshed 3 minutes ago

Figure: The green taxi data

- whereas the yellow taxi parquet file has 663,055,251 rows and 19 columns.

```
The number of rows in yellow taxi parquit file are: 146039231
The number of columns present in the yellow taxi parquet: 19
```

Figure: Displaying number of rows and columns of yellow parquet

Table +

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID
1	1	2015-01-01T00:11:33.000+0000	2015-01-01T00:16:48.000+0000	1	1	1
2	1	2015-01-01T00:18:24.000+0000	2015-01-01T00:24:20.000+0000	1	0.9	1
3	1	2015-01-01T00:26:19.000+0000	2015-01-01T00:41:06.000+0000	1	3.5	1
4	1	2015-01-01T00:45:26.000+0000	2015-01-01T00:53:20.000+0000	1	2.1	1
5	1	2015-01-01T00:59:21.000+0000	2015-01-01T01:05:24.000+0000	1	1	1
6	1	2015-01-01T00:07:31.000+0000	2015-01-01T00:11:32.000+0000	1	0.8	1
7	1	2015-01-01T00:47:08.000+0000	2015-01-01T00:54:50.000+0000	1	1.1	1

10,000 rows | Truncated data | 50.78 seconds runtime Refreshed now

Figure: Yellow taxi data

- **Answer to the question 5** :The size of the green taxi 2015 parquet file is 404556105 bytes, but when it is converted into csv the size of the file is 0 bytes.

- The location referential csv has the locations as shown below:

Table +

	LocationID	Borough	Zone	service_zone
1	1	EWR	Newark Airport	EWR
2	2	Queens	Jamaica Bay	Boro Zone
3	3	Bronx	Allerton/Pelham Gardens	Boro Zone
4	4	Manhattan	Alphabet City	Yellow Zone
5	5	Staten Island	Arden Heights	Boro Zone
6	6	Staten Island	Arrochar/Fort Wadsworth	Boro Zone
7	7	Queens	Astoria	Boro Zone

265 rows | 1.44 seconds runtime

Figure: Refrential locations

ARCHITECTURE:

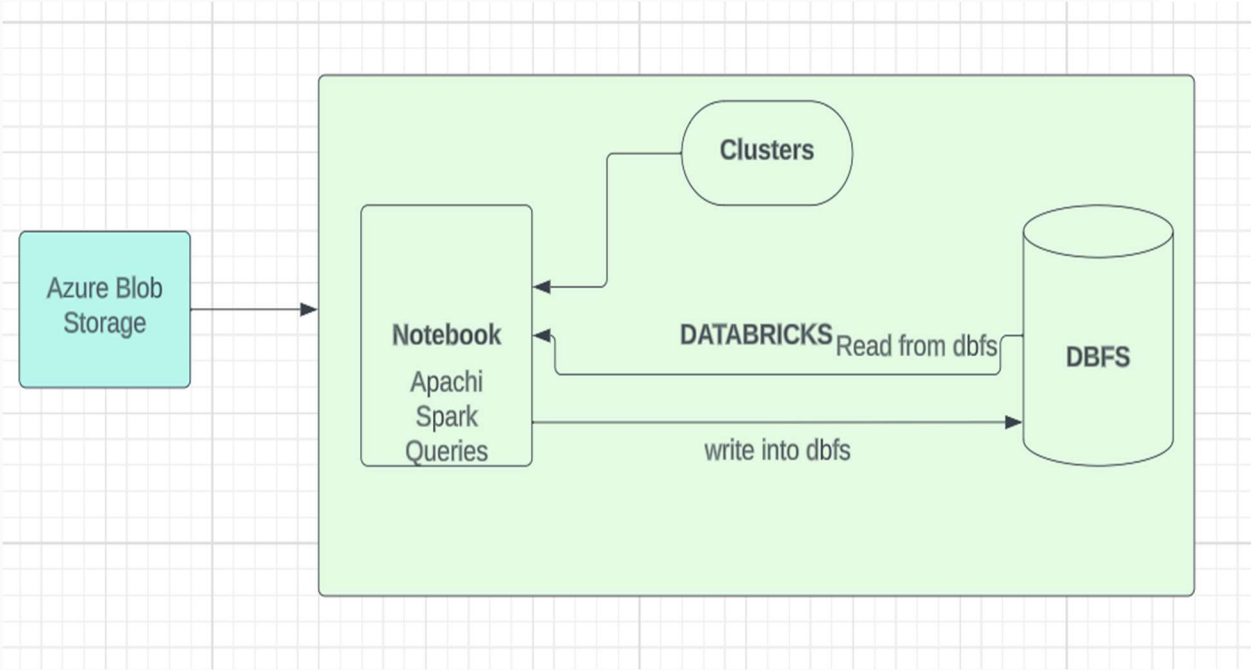


Figure : Architecture

Data cleaning to remove unrealistic trips records of green and yellow taxi.

a. Trips finishing before the starting time

The green and yellow taxi parquet files have data that is unrealistic in nature, such as the date and time of the trip, which show that the trip has ended before it has started, which is wrong; unless the trip has started, it cannot be ended. So the filter function taxi.filter(F.col("pep_dropoff_datetime") > F.col("pep_pickup_datetime")) helped in removing the unrealistic trip data where the drop-off time is earlier than the pick-up time.

b. Trips where the pickup/dropoff datetime is outside of the range.

The taxi drivers make trips anytime during the clock; the trip data should be any time from 0 to 24 hours, but the yellow taxi and green taxi files contain data

that is beyond 24 hours, which is unrealistic. By defining the range as start_time="2015-01-01T00:00:00.000+0000" and end_time="2015-12-31T23:59:59.000+0000" the stored trip time that falls outside of the given range gets removed with the help of the filter function.

c. Trips with negative speed

The speed of the taxi cannot be below 0, as it won't run at 0 mph. To calculate the trip duration in hours, first a column is created named "trip_duration_hours," then the trip pickup and drop-off duration time is converted into hours. After this, another column, "trip_speed_mph", is created to store the results obtained by the trip speed calculation in miles per hour by dividing the trip distance by the trip duration in hours. Finally, trips with negative speeds are filtered out with the help of a filter function to remove unrealistic data.

d.Trips with very high speed (look for NYC and outside of NYC speed limit)

The NYC speed limit is 55 kmph, and the speed limit outside of NYC is 75 kmph unless it is specified on the roadside. A taxi records its GPS coordinates and speed during a trip. Due to a malfunction in the GPS device or a data recording error, the recorded speed of a trip can be less than 0 miles per hour, which means the taxi has moved backward during the trip. This is an unrealistic scenario. This data can be filtered by defining a speed limit between 0 and 75. The trip record that shows a speed limit less than 0 and greater than 75 is dropped with the filter function.

e.Trips that are travelling too short or too long (duration wise)

The trip can be too short; it can last only a few minutes and also be continued for hours. The fare of the taxi trip starts immediately when the taxi starts moving with the passenger on board, and in some taxis, the fare starts as soon as the taxi's metre is activated. The scenarios where people prefer to take a short trip are: The first scenario to take into account is that elderly people who cannot walk continuously for 10 to 15 minutes to reach the station will take a taxi to reach the station, and as the distance is shorter, the taxi can drop them off within 2 minutes. The other scenario can be people who have gone for grocery shopping; even if the distance is too short, it's difficult for them to walk with heavy baggage. Beside this, taxi drivers are restricted from driving continuously beyond 12 hours. Considering these scenarios, the trip data that is below 1.5 minutes and above 12 hours is dropped with the filter function.

f.Trips that are travelling too short or too long (distance wise)

The shortest distance can be less than 1 kilometre, or 0.5 miles, depending on the requirements of the passenger. The passenger does not prefer to take a taxi for less than half a kilometre unless there is an emergency, and the situation demands it. There are also certain restrictions on taxi drivers: they should not drive beyond the distance specified by the government rules, as driving for too long continuously may make the driver tired, and the engine of the taxi must be cooled down to avoid accidents. Considering the limit of too short and too long trips as 0.5 miles to 180 miles, the trips that do not fall under the limit are filtered out using filter((F.col("trip_distance") >= min_distance_miles) &(F.col("trip_distance") <= max_distance_miles)) as they come under unrealistic trips.

f.Any other logic you think is important.

- The taxi trip without a passenger cannot be considered a trip; the unrealistic trip record of 0 passengers is removed.
- The yellow taxi files and green taxi files have similar types of records. To merge all the records into one dataframe, the columns named "lpep_pickup_datetime" , "tpep_pickup_datetime", "lpep_dropoff_datetime" and "tpep_dropoff_datetime" are renamed pickup and drop-off with the help of the function rename("lpep_pickup_datetime" , "pickup_datetime").withColumnRenamed("lpep_dropoff_datetime", "dropoff_datetime")
- The record of trip data such as passenger count, RatecodeId, improvement surcharge, and congestion surcharge has missing values, which are replaced using the fillna() function, and the missing values are filled with 0.
- The extra columns that are created during the filtering of data are dropped.
- The column payment_type should be in double format, as the payment can be in decimals, so it is typecast with a double data type.
- To get accurate records of trips, the pickup and drop-off locations with reference to location IDs are extracted from the referential location and joined into yellow and green parquet file data. All the data for both yellow and taxis is "unionByName" to bring all the cleaned data together.
- The merged data is written into bdfs.
- At the end the spark session has to be terminated.

Figure: Combined data of yellow and green taxi

PART 2: DATA ANALYSIS:

Temporary view of cleaned data is created to perform queries.
The functions of spark helps in extracting the desired information.

1. For each year and month (e.g January 2020 => "2020-01-01" or "2020-01" or "Jan 2020":

1. What was the total number of trips?
2. Which day of week (e.g. monday, tuesday, etc..) had the most trips?
3. Which hour of the day had the most trips?
4. What was the average number of passengers?
5. What was the average amount paid per trip (using *total_amount*)?
6. What was the average amount paid per passenger (using *total_amount*)?

The WITH CTE (Ranked_yearmonth) is used to calculate the total number of trips, the day with the most trips, the hour of the day that had the most trips, the average number of passengers, the average amount paid per trip, and the average amount paid per passenger. The above calculation is partitioned by year_month using the ROW_NUMBER() OVER PARTION function. The DATE_FORMAT method is used to get the date in year-month format. The COUNT function is used to count the total trips, and the DATE_FORMAT(DATETIME, 'EEEE') function is used to calculate the day with the most trips. The HOUR method is used to find the hour with the most trips. The AVG function is used to find the average number of passengers, the average amount per trip, and the average amount per passenger, and then the ROUND method is used to get the output in two decimal places. Finally, group the results by year_month, most_trips_day_of_week, and most_trips_hour_of_day to get the desired output.

All the columns created in the WITH CTE are called in the SELECT statement, and the WHERE condition with rank=1 is applied to get the most earned trips. The order by clause is used to display the result in ascending order of year_month.

Table +

	year_month	total_trips	most_trips_day_of_week	most_trips_hour_of_day	avg_passengers	avg_amount_per_trip
1	2015-01	165656	Friday	19	1.67	14.95
2	2015-02	137012	Friday	19	1.64	15.94
3	2015-03	157895	Sunday	0	1.72	15.7
4	2015-04	157677	Thursday	19	1.64	16.09
5	2015-05	156699	Friday	19	1.67	16.03
6	2015-06	137494	Tuesday	19	1.63	15.72
7	2015-07	138248	Wednesday	19	1.64	15.74

96 rows | 13.40 minutes runtime Refreshed 2 days ago

Figure : Query 1

2. For each taxi colour (yellow and green):
a.What was the average, median, minimum and maximum trip duration in minutes (with 2 decimals, eg. 90 seconds = 1.50 min)?
b.What was the average, median, minimum and maximum trip distance in **km**?
What was the average, median, minimum and maximum speed in **km per hour**?

The green and yellow taxi trips are indicated by their colours, i.e., yellow and green. The group by clause is used to segregate the calculations, such as average, median, minimum and maximum of trip duration, trip distance, and speed, based on the taxi colour, such as yellow and green. To make the calculation easier, a trip duration column was added after reading the final cleaned data.

Before applying the AVG, PERCENTILE_APPROX (MEDIAN), MIN, and MAX functions to the trip duration, the trip duration is divided by 60 to get its corresponding values in minutes. Before applying the AVG, PERCENTILE_APPROX (MEDIAN), MIN, and MAX functions to the trip distance, the trip distance is multiplied by 1.60934 to get its corresponding values in km. Similarly, before calculating the speed, the trip distance is multiplied by 1.60934, and the trip distance is multiplied by 0.000277778 to get the speed in km

per hour. Finally, AVG, PERCENTILE_APPROX (MEDIAN), MIN, and MAX functions are applied to speed to get its corresponding values.

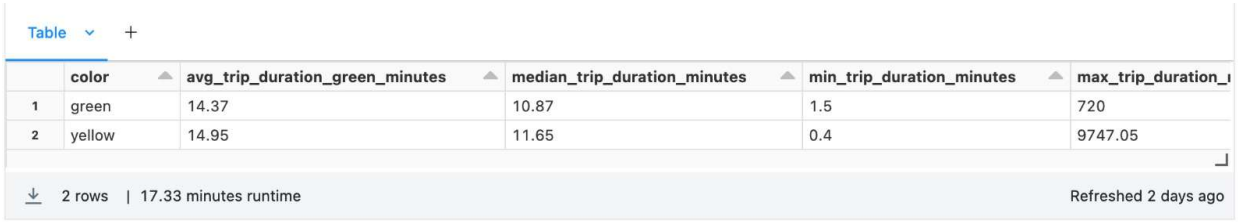


Table +

	color	avg_trip_duration_green_minutes	median_trip_duration_minutes	min_trip_duration_minutes	max_trip_duration_
1	green	14.37	10.87	1.5	720
2	yellow	14.95	11.65	0.4	9747.05

2 rows | 17.33 minutes runtime Refreshed 2 days ago

Figure : Query 2

3. **For each taxi colour (yellow and green), each pair of pick up and drop off locations (use boroughs not the id), each month, each day of week and each hours:**
    1. What was the total number of trips?
    2. What was the average distance?
    3. What was the average amount paid per trip (using *total_amount*)?
    4. What was the total amount paid (using *total_amount*)?

The group by clause is used to segregate the results based on taxi colour, borough pickup location, borough drop-off location, trip month, trip day of the week, and trip hour. The MONTH, DATE_FORMATE(DATETIME,'EEEE'), and HOUR methods are applied to get the trip month, trip day of the week, and trip hour on the pickup date. The COUNT method is used to calculate the total trips. The AVG, SUM, and ROUND methods are used to calculate the average distance, average amount per trip, and total amount paid in two decimal places.

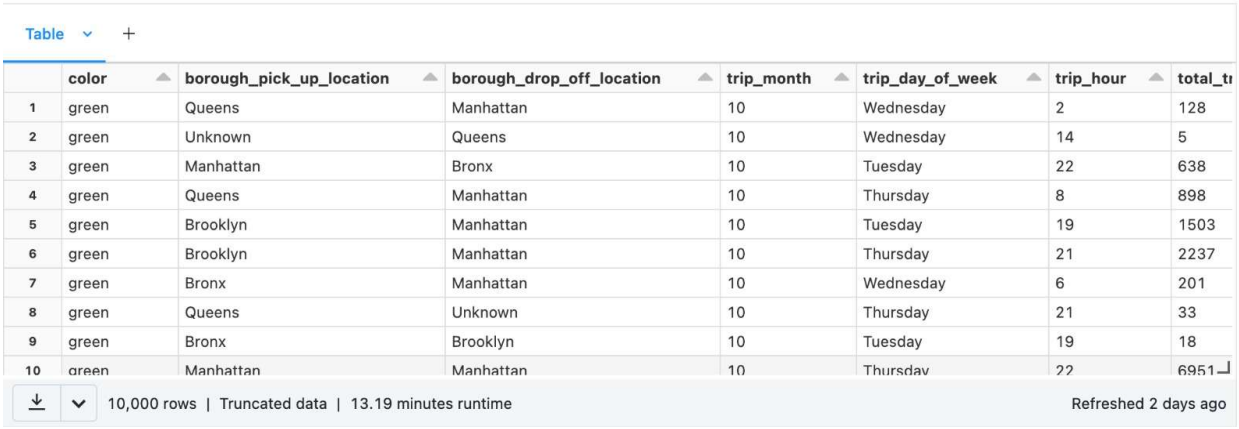


Table +

	color	borough_pick_up_location	borough_drop_off_location	trip_month	trip_day_of_week	trip_hour	total_tr
1	green	Queens	Manhattan	10	Wednesday	2	128
2	green	Unknown	Queens	10	Wednesday	14	5
3	green	Manhattan	Bronx	10	Tuesday	22	638
4	green	Queens	Manhattan	10	Thursday	8	898
5	green	Brooklyn	Manhattan	10	Tuesday	19	1503
6	green	Brooklyn	Manhattan	10	Thursday	21	2237
7	green	Bronx	Manhattan	10	Wednesday	6	201
8	green	Queens	Unknown	10	Thursday	21	33
9	green	Bronx	Brooklyn	10	Tuesday	19	18
10	green	Manhattan	Manhattan	10	Thursday	22	6951

10,000 rows | Truncated data | 13.19 minutes runtime Refreshed 2 days ago

Figure :Query result on each pair of pick up and drop off locations

4. What was the percentage of trips where drivers received tips?

Ans) WITH CTE (tip_trips) is used to calculate the count of trips with tips, and one more CTE (all_trips) is used to calculate the count of all trips. Finally, the CROSS JOIN and ROUND method is used to calculate the percentage of trips where drivers received tips in two decimal places.

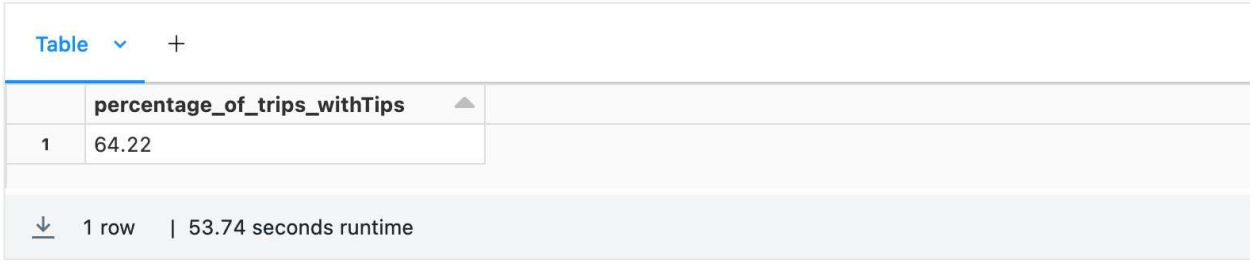


Table +

	percentage_of_trips_withTips
1	64.22

1 row | 53.74 seconds runtime

Figure : percentage of trips where drivers received tips

5. For trips where the driver received tips, what was the percentage where the driver received tips of at least $5.

WITH CTE (received_tips) is used to calculate the count of trips with tips amount greater than or equal to $5.00, and one more CTE (all_trips) is used to calculate the count of all trips. Finally, the CROSS JOIN and ROUND method is used to calculate the percentage of trips where drivers received tips of at least $5.00 in two decimal places.

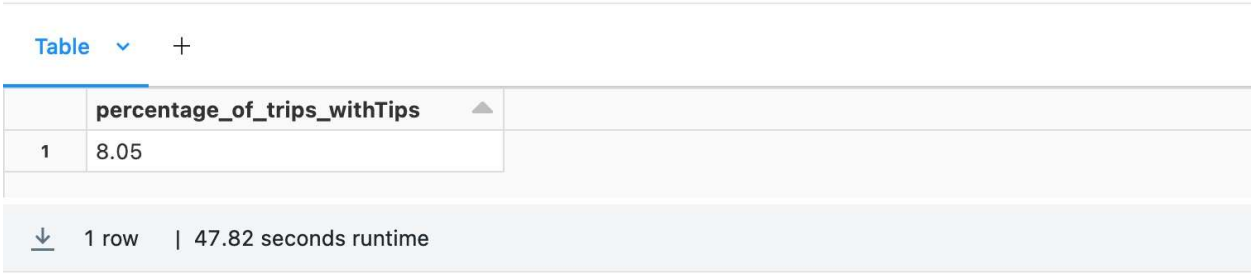


Figure : percentage where the driver received tips of at least $5.

6. Classify each trip into bins of durations:

1. **Under 5 Mins**
2. **From 5 mins to 10 mins**
3. **From 10 mins to 20 mins**
4. **From 20 mins to 30 mins**
5. **From 30 mins to 60 mins**
6. **At least 60 mins**

Then for each bins, calculate:

a. Average speed (km per hour)

b. Average distance per dollar (km per $)

To get the trip speed in kmph, the trip distance is multiplied by 1.60934, and its output is divided by the result of the trip duration multiplied by 0.000277778.

The WITH CTE (bins) is used to calculate the average speed in kmph and the average distance per dollar. The AVG method is used to calculate the average, and the ROUND method is used to calculate the average in two decimal places. The CASE WHEN is used to calculate the bins based on the trip duration in minutes. The group-by clause is used to segregate the outcomes based on their respective bins. The order-by clause is used to display the results in descending order.

Table +

	bins	average_speed_kmph	average_distance_per_dollar
1	Under 5 Mins	20.98	0.26
2	At least 60 Mins	22.24	2.81
3	5 to 10 Mins	17.21	0.29
4	30 to 60 Mins	25.64	0.71
5	20 to 30 Mins	21.18	0.42
6	10 to 20 Mins	17.72	0.34

6 rows | 5.99 minutes runtime Refreshed 2 days ago

Figure :Query 6 :Bins on distance in minutes

Question 7. Which duration bin will you advise a taxi driver to target to maximize his income?

You would need to take into account a variety of elements, such as the local market conditions and the driver's strategy, in order to advise a taxi driver on the duration bin to target in order to maximise his income. The bins are selected based on:
Short Distance Trips (Under 5 and 5 to 10 Minutes): Due to the tiny distances they traverse, these journeys are typically low paying.

Due to their brief lifespan, they might offer a high turnover, but they might not be the most profitable. In general, it's a good idea to go for longer routes with higher fares.

- Trips of Medium Length (10–20 Min. and 20–30 Min.):

  These excursions may provide a healthy balance between turnover and revenue.They travel fair distances and frequently offer affordable fares.These time bins can be targeted to bring in a consistent flow of customers and revenue.

- Long Distance Trips (at least 60 minutes and 30 to 60 minutes): Given that they cover significant distances, long trips may have higher potential fares.They might, however, have longer waits in between trips, so it's important to weigh the overall compensation for the time spent.

  If there is a demand, particularly during busy times or for special events, targeting long-duration journeys may be financially advantageous.

In the end, factors including the local market demand, the time of day, and particular driver preferences may affect the best duration bin for maximising revenue. During times of high demand, a well-balanced strategy can include a combination of medium- and long-duration journeys, with shorter trips filling in the gaps. In order to maximise income, it is crucial for the driver to keep an eye on their earnings and modify their plan as necessary.

PART 3: Machine Learning

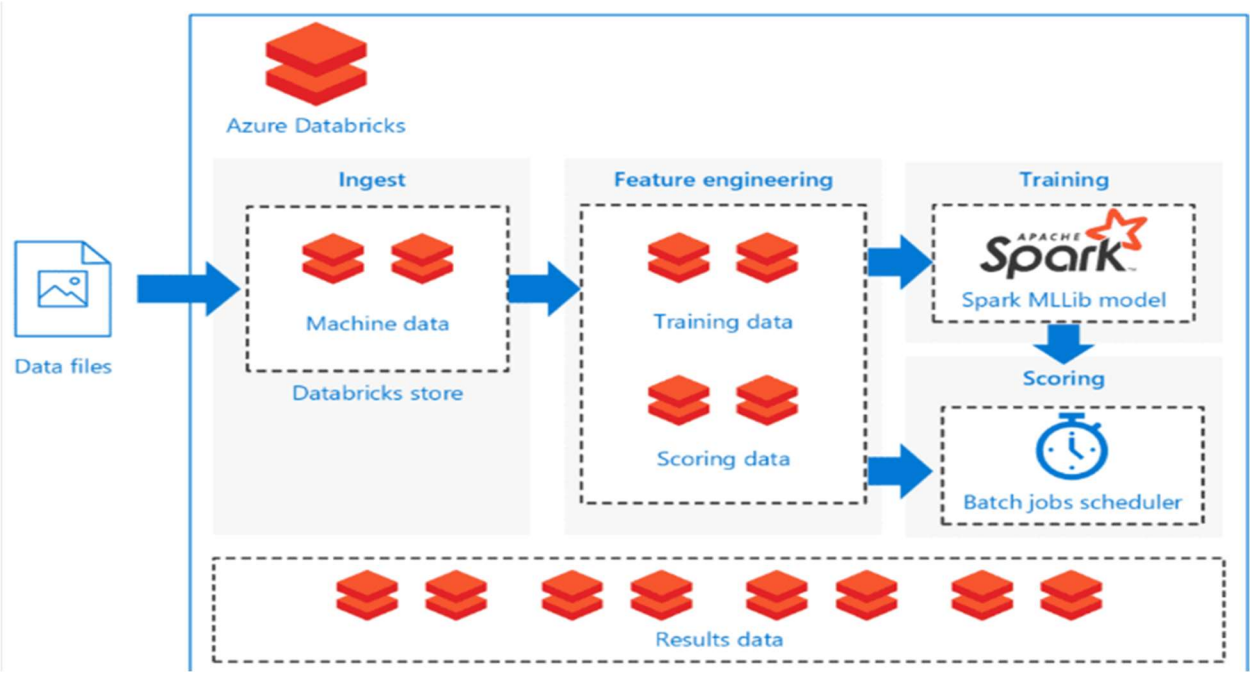


Figure: Machine learning modelling process

Baseline model on average paid per trip

The baseline is the dumb model created to know the actual performance of the data. Here baseline is calculated by multiplying the target values with mean as the model is based on regression supervised machine learning. The mean square error of the baseline model is 12.053699. This value will be compared with model predicted values to know whether the model beats this dumb model or not. The run time of the model is 6.91 seconds.

- **Use all data except October/NovemberDecember 2022 to train and validate your models and use the RMSE score to assess your models.**

- The categorical column “"store_and_fwd_flag" is converted into binary using indexer and onehot encoding then all the numerical, categorical data is fitted into pileline.
- Then features and labels as assigned.
- The data is scaled to normalize the data and to avoid biasness using Standard Scaler function.
- The data is split into train, validation and test in the ration of 0.6, 0.2,0.2. The train data is used to fit the data into model and predict the accuracy of model. Validation data is used to analyze whether the model is accurately reading the data or not. As the train data is already read by model, the unseen data is passed to validate the performance of the model.
- Test data helps in evaluating the performance of model.
- The matrix mean square error of Regression Evaluator helps in analyzing the performance of model and it helps in knowing whether the data is underfitted, or overfitted or model predicts well.
- **Baseline score : 737.7268518845964**

Model 1:

Linear Regression:

train_mse score: 254.785
val_mse score: 240.500
test_mse score: 256.9639

Validation set underfits and test set overfits the data

Model 2:

Support vector machine:

train_mse score:463.359
val_mse score: 548.878
test_mse score: 234.999

Validation and test data underfits the model

Choose your best model and explain why you choose it (processing time, complexity, accuracy, etc).

The analysis on two models gives an insight that model 1 gives the best accuracy score compared to model 2. The model 1 beats the baseline model. The processing time for each model was beyond two hours, cluster was breaking each time the model was on execution process.

Using your best model, predict the October/NovemberDecember 2022 trips and calculate the RMSE on your predictions. Does your model beat the baseline model

Baseline score: 1325.256573782342
Train_mse score:853.359
val_mse score: 748.878
test_mse score: 834.999

Yes, this model beats the baseline model

Any issues/bugs you faced and how you solved them.

1. **Problem:** Notebook getting detached

   Solution: In data cleaning process, due to garbage collection, notebook was getting detached as the cluster was not able to solve the problem with incorrect conditions.

2. **Problem:** Notebook not able to export

   Solution: While running multiple codes into one single notebook, the size of the notebook was getting exceeded the limit size of the notebook, so to deal with this issue, the code is executed into separate notebooks. The green taxi data is transformed in one notebook and was written into dbfs, yellow taxi data was transformed separate notebook. To merge both the data, green taxi data is read into the yellow taxi notenook and then both data is combined and written into the dbfs.

3. **Problem:** The location referential csv when merged with yellow taxi data and green taxi data was getting converted into object type.

   Solution: The imported location referential csv is written into dbfs then read to combine borough into taxi data.

4. **Problem:** Data modelling. As the data was huge, the execution time was taking long time, and there were several jobs and cluster was running in the loop.
   Solution: Sample of data from 2020 to 2022 is taken to make models.

References:

1 .Data processing on a big dataset with Databricks Spark and azure - Bing images

Trip duration limit:

2. Passenger Frequently Asked Questions - TLC (nyc.gov)
3. CSV file | Databricks on AWS
4. read-parquet-files - Databricks (microsoft.com)
5. Techniques for Cleaning and Preprocessing Data in Apache Spark Dataframes (sparkcodehub.com)
6. Parquet Files - Spark 3.5.0 Documentation (apache.org)
7. https://spark.apache.org/mllib/
8. https://spark.apache.org/docs/latest/ml-classification-regression.html#linear-support-vector-machine