Big Data Engineering - Assignment 2: Data Processing

Ronik Jayakumar
24680264
ronik.jayakumar@student.uts.edu.au

1.0 Introduction

The project deals with the usage of Apache Spark on the DataBricks platform to analyse The New York City Taxi and Limousine Commission (TLC) datasets from the year 2015 to 2022. This includes data from the green taxis (which are taxis that respond to street hails, but only in certain designated areas) and yellow taxis (the taxis from New York city that have the right to pick up street-hailing passengers anywhere in the city).

The project includes 3 major sections: Data Ingestion & Preparation, Business Questions, and Machine Learning. The 3 sections are discussed in detail in this report.

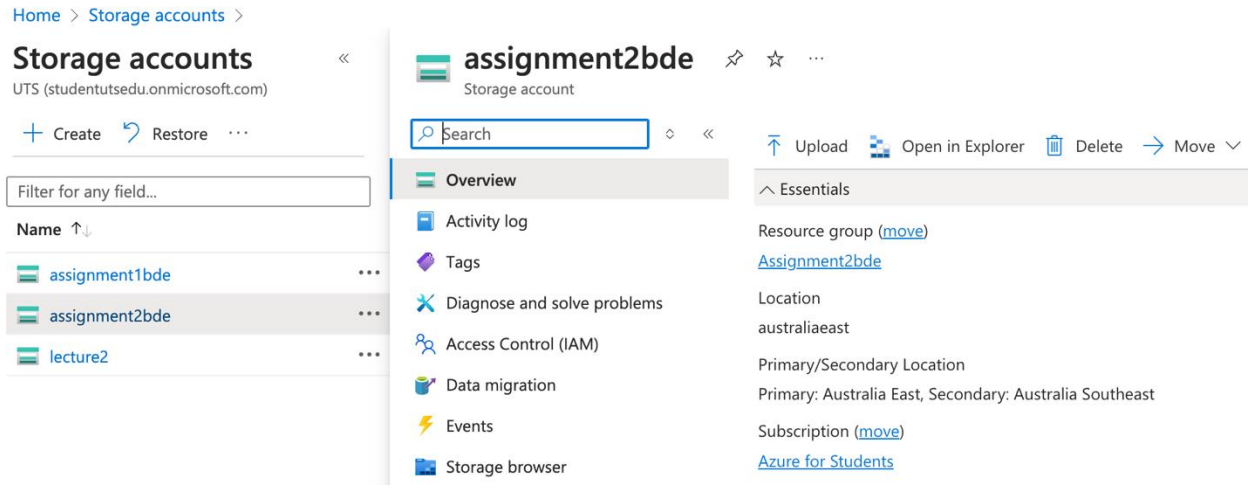
2.0 Data Ingestion and Preparation

The New York City Taxi Data has been used in this project which has been downloaded off the provided Google Drive Link. The data is provided year-wise for each taxi colour (green and yellow).

Some of the downloaded files exceed the 2-gigabyte upload limit on the file upload system of DataBricks. Due to this reason, the files were first uploaded onto Microsoft Azure, following which the unique key was used to ingest this data into the DataBricks File System or DBFS.

2.1 Microsoft Azure

Create a container within a storage account on Microsoft Azure where all the 16 files are uploaded (2015 - 2022 data for both green and yellow taxis)



2.2 Import the Datasets onto DataBricks

The uploaded files are imported from Azure onto the DataBricks File Storage System using the following steps.

Step 1: Defining the storage accounts

```
9/12/2024 (<1s)                    3: Connect Azure Database

storage_account_name = "assignment2bde"
storage_account_access_key = "JWpE7mR2VwLc2P8roETpnvXt7Awu14jy/NcKILnHqIniK904xf9UsKxlVtXrXnPeulWwXAUPcEFM+ASteAcTBg=="
blob_container_name = "taxidata"
```

Step 2: Import the files onto DBFS

```
4

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}
)
```

```
5                                                    Python

azure_green_path = f"/mnt/taxidata/Green"
azure_yellow_path = f"/mnt/taxidata/Yellow"
dbfs_green_path = "/dbfs/mnt_1/taxidata/Green"
dbfs_yellow_path = "/dbfs/mnt_1/taxidata/Yellow"

# Copy files from Azure to DBFS
dbutils.fs.cp(azure_green_path, dbfs_green_path, recurse=True)
dbutils.fs.cp(azure_yellow_path, dbfs_yellow_path, recurse=True)
```

2.3 Load and Count the Data

The data was loaded and saved onto the local DBFS. The loaded data files were merged according to their corresponding taxi colour to verify the count of rows in each

```
9/26/2024 (2s)                                       11

# Re-read the files from the local DBFS
green_taxi_files = [file.path for file in dbutils.fs.ls("/dbfs/mnt_1/taxidata/Green")]
green_taxi_df = spark.read.parquet(*green_taxi_files)

yellow_taxi_files = [file.path for file in dbutils.fs.ls("/dbfs/mnt_1/taxidata/Yellow")]
yellow_taxi_df = spark.read.parquet(*yellow_taxi_files)

(2) Spark Jobs
green_taxi_df:  pyspark.sql.dataframe.DataFrame = [VendorID: long, lpep_pickup_datetime: timestamp ... 18 more fields]
yellow_taxi_df:  pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime: timestamp ... 17 more fields]
```

```
9/23/2024 (22s)                                                    12

green_taxi_row_count = green_taxi_df.count()
print(f"Total rows in green taxi datasets: {green_taxi_row_count}")

yellow_taxi_row_count = yellow_taxi_df.count()
print(f"Total rows in yellow taxi datasets: {yellow_taxi_row_count}")

▸ (4) Spark Jobs

Total rows in green taxi datasets: 66200401
Total rows in yellow taxi datasets: 663055251
```

2.4 Parquet vs. CSV

As a comparison parameter, the green taxi data from 2015 is converted to a comma separated value file to check the size difference between the two formats. The results show the size of CSV file to be almost 1.8 times larger than the size of the parquet file.

```
9/12/2024 (1s)                                                     19

parquet_file = dbutils.fs.ls("/mnt/taxidata/green_taxi_2015.parquet")[0]
parquet_file_size_mb = parquet_file.size / (1024 * 1024)
print(f"Parquet File Size: {parquet_file_size_mb:.2f} MB")

csv_file = dbutils.fs.ls(f"/mnt/taxidata/green_taxi_2015.csv")[0]
csv_file_size_mb = csv_file.size / (1024 * 1024)
print(f"CSV File Size: {csv_file_size_mb:.2f} MB")

Parquet File Size: 385.81 MB
CSV File Size: 685.02 MB
```

2.5 Data Preprocessing

The parquet data files undergo a series of preprocessing steps to ensure they are ready for analysis and modelling. This involves removing faulty data within the DataFrame which could give us unrealistic results. The preprocessing steps are highlighted below step by step:

1. Remove rows where the passenger drop-off time is before the pickup time.

```
4 days ago (9s)                                                    5
yellow_taxi_df.createOrReplaceTempView('yellow_taxi')
yellow_taxi_df = spark.sql("""
                        SELECT *
                        FROM yellow_taxi
                        WHERE tpep_dropoff_datetime > tpep_pickup_datetime
                        """)
```

```
yellow_taxi_df:  pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime: timestamp ... 17 more fields]
```

```
4 days ago (<1s)                                                   6
green_taxi_df.createOrReplaceTempView('green_taxi')
green_taxi_df = spark.sql("""
                        SELECT *
                        FROM green_taxi
                        WHERE lpep_dropoff_datetime > lpep_pickup_datetime
                        """)
```

```
green_taxi_df:  pyspark.sql.dataframe.DataFrame = [VendorID: long, lpep_pickup_datetime: timestamp ... 18 more fields]
```

2. Remove any rows that are out of the defined date range (2015 to 2022)

```
4 days ago (1s)                                                    8
green_taxi_df.createOrReplaceTempView('green_taxi')
green_taxi_df = spark.sql("""
                        SELECT *
                        FROM green_taxi
                        WHERE lpep_pickup_datetime >= '2015-01-01T00:00:00'
                        AND lpep_pickup_datetime <= '2022-12-31T23:59:59'
                        """)
```

```
green_taxi_df:  pyspark.sql.dataframe.DataFrame = [VendorID: long, lpep_pickup_datetime: timestamp ... 18 more fields]
```

```
4 days ago (<1s)                                                   9
yellow_taxi_df.createOrReplaceTempView('yellow_taxi')
yellow_taxi_df = spark.sql("""
                        SELECT *
                        FROM yellow_taxi
                        WHERE tpep_pickup_datetime >= '2015-01-01T00:00:00'
                        AND tpep_pickup_datetime <= '2022-12-31T23:59:59'
                        """)
```

```
yellow_taxi_df:  pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime: timestamp ... 17 more fields]
```

3. Filter out all the trips with negative speed. To calculate the speed, the trip duration is calculated using the pickup and drop off times. The trip distance is converted to kilometers so the speed can be calculated using the kilometer per hour metric. This is executed as given below.
    a. Calculate the metrics

```
4 days ago (1s)                                                    11                                                    Python

green_taxi_df = green_taxi_df.withColumn('trip_duration_hours',
                    round((unix_timestamp(col("lpep_dropoff_datetime")) - unix_timestamp(col("lpep_pickup_datetime"))) / 3600, 2))

green_taxi_df = green_taxi_df.withColumn('trip_distance_km',
                    round(col("trip_distance") * 1.6093, 2))

green_taxi_df = green_taxi_df.withColumn("speed_kmph",
                    round(col("trip_distance_km") / col("trip_duration_hours"), 2))

▸ green_taxi_df:  pyspark.sql.dataframe.DataFrame = [VendorID: long, lpep_pickup_datetime: timestamp ... 21 more fields]
```

```
4 days ago (<1s)                                                   12

yellow_taxi_df = yellow_taxi_df.withColumn('trip_duration_hours',
                    round((unix_timestamp(col("tpep_dropoff_datetime")) - unix_timestamp(col("tpep_pickup_datetime"))) / 3600, 2))

yellow_taxi_df = yellow_taxi_df.withColumn('trip_distance_km',
                    round(col("trip_distance") * 1.6093, 2))

yellow_taxi_df = yellow_taxi_df.withColumn("speed_kmph",
                    round(col("trip_distance_km") / col("trip_duration_hours"), 2))

▸ yellow_taxi_df:  pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime: timestamp ... 20 more fields]
```

b. Filter out the negative speed

```
4 days ago (<1s)                                                   13

green_taxi_df.createOrReplaceTempView('green_taxi')
green_taxi_df = spark.sql("""
                    SELECT *
                    FROM green_taxi
                    WHERE speed_kmph  > 0
                    """)

▸ green_taxi_df:  pyspark.sql.dataframe.DataFrame = [VendorID: long, lpep_pickup_datetime: timestamp ... 21 more fields
```

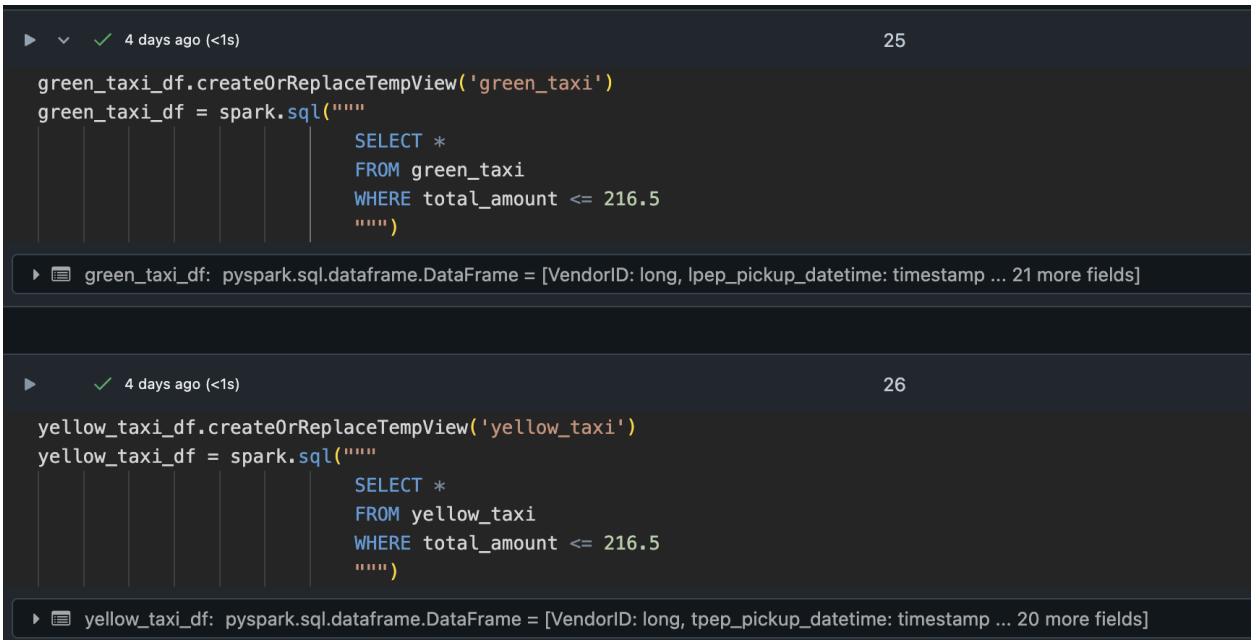
```
4 days ago (<1s)                                                   14

yellow_taxi_df.createOrReplaceTempView('yellow_taxi')
yellow_taxi_df = spark.sql("""
                    SELECT *
                    FROM yellow_taxi
                    WHERE speed_kmph  > 0
                    """)

▸ yellow_taxi_df:  pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime: timestamp ... 20 more field
```

4. The New York City wide speed limit has been imposed as 25 miles per hour unless stated otherwise. After referring to multiple sources and articles, a speed limit of 35mph has been considered. [1], [2]

5. The records with exceedingly low or large trip durations have been filtered out. The minimum trip duration has been considered as 2 minutes and the maximum has been considered as 2.5 hours as according to google maps, the time taken to travel between to 2 furthest boroughs in the dataset, i.e. Staten Island and the Bronx during rush hour is approximately 2 hours and 10 minutes. An extra 20 minutes leeway is considered.

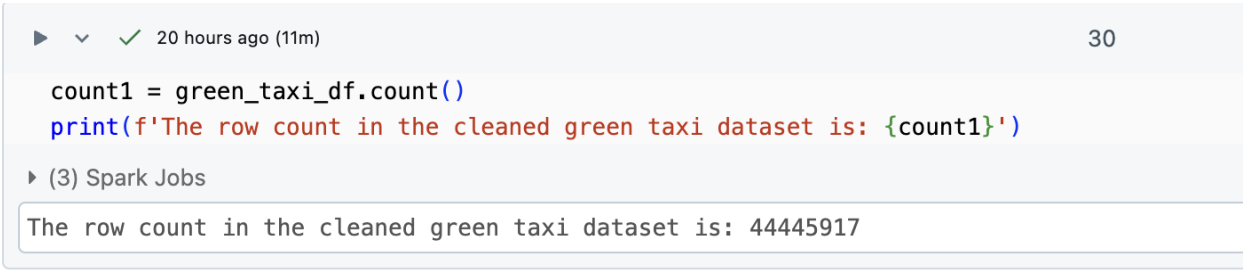
6. Similarly, the records with extremely low or large trip distances have been filtered out. The minimum and maximum trip distances are considered as 2km and 75kms respectively. The maximum distance has been calculated considering the distance between Staten Island and the Bronx being approximately 60kms.

7. Trips with exceedingly high total amounts have been dropped. A maximum total amount of $216.5 has been considered as the maximum cost to travel from Staten Island to the Bronx can be $210. Taking into account tax, tips, tolls, and other factors, an extra $10 cushion has been considered.



8. Any duplicate rows within the dataframe have been dropped.

Following these preprocessing steps, the final row count of the green and yellow taxi datasets were as follows:



This indicates **21,754,494** rows removed from Green taxi data and **239,064,477** rows removed from the yellow taxi data.

After row wise filtration, the columns in the two DataFrames were analysed to ensure similar columns exist in both datasets. To reduce the overall size of the datasets, the columns with lesser significance to our business case were dropped from further processing. These columns for the yellow and green taxi datasets were as follows:

```
5 days ago (<1s)                                                    33
yellow_taxi_df = yellow_taxi_df.drop('payment_type', 'RatecodeID', 'store_and_fwd_flag', 'trip_distance', 'VendorID', 'improvement_surcharge',
'mta_tax', 'airport_fee', 'congestion_surcharge', 'tolls_amount')
yellow_taxi_df:  pyspark.sql.dataframe.DataFrame = [tpep_pickup_datetime: timestamp, tpep_dropoff_datetime: timestamp ... 10 more fields]
+ Code   + Text
5 days ago (<1s)                                                    34
green_taxi_df = green_taxi_df.drop('store_and_fwd_flag', 'RatecodeID', 'ehail_fee', 'trip_type', 'congestion_surcharge', 'payment_type',
'trip_distance', 'VendorID', 'improvement_surcharge', 'mta_tax', 'tolls_amount')
green_taxi_df:  pyspark.sql.dataframe.DataFrame = [lpep_pickup_datetime: timestamp, lpep_dropoff_datetime: timestamp ... 10 more fields]
```

The two dataframes are then merged into a single taxi dataframe.

Following the merge, the reference table is joined with the merged taxi dataset, joining the borough data with the pickup and drop-off location ID's as given in the merged dataset.

This merged dataset is saved as a parquet file and also as a table for future querying use.

```
4 days ago (42s)                                                    52
spark.sql("DROP TABLE IF EXISTS taxi_database.taxi_table")
spark.sql("DROP DATABASE IF EXISTS taxi_database")
spark.sql("CREATE DATABASE taxi_database")
spark.sql("USE taxi_database")
spark.sql("CREATE TABLE taxi_table USING PARQUET OPTIONS (path '/dbfs/mnt_1/taxidata/taxi_combined.parquet')")
(1) Spark Jobs
```

3.0 Business Questions

3.1 Business Question 1

1. For each year and month (e.g January 2020 => "2020-01-01" or "2020-01" or "Jan 2020":
    a. What was the total number of trips ?
    b. Which day of week (e.g. monday, tuesday, etc..) had the most trips ?
    c. Which hour of the day had the most trips ?
    d. What was the average number of passengers ?
    e. What was the average amount paid per trip (using *total_amount*) ?
    f. What was the average amount paid per passenger (using *total_amount*) ?

=> In a Single table/dataframe/output

The result of business question 1 consists of the total trips for each month, the day of the week and the hour of the day with the highest number of trips, the average passenger count rounded off to a whole number, the average total amount per trip and the average price paid per passenger.

	year_month	total_trips	most_trips_day_of_week	most_trips_hour_...	average_passenger_co...	average_total_amount	average_amount_per_passenger
1	2015-01	8875952	Saturday	7PM	2	17.72	14.44
2	2015-02	8753784	Saturday	7PM	2	18.5	15.14
3	2015-03	9572483	Sunday	9PM	2	18.92	15.47
4	2015-04	9530635	Thursday	9PM	2	19.03	15.5
5	2015-05	9853447	Saturday	9PM	2	19.47	15.8
6	2015-06	9086315	Tuesday	9PM	2	19.47	15.83
7	2015-07	8580684	Thursday	10PM	2	19.06	15.44
8	2015-08	8352999	Saturday	9PM	2	19.08	15.43
9	2015-09	8283875	Wednesday	9PM	2	19.51	15.86
10	2015-10	8996993	Saturday	7PM	2	19.71	16.05
11	2015-11	8232090	Sunday	7PM	2	19.43	15.84
12	2015-12	8298644	Thursday	9PM	2	19.5	15.82
13	2016-01	7712879	Friday	6PM	2	18.76	15.34
14	2016-02	8010301	Saturday	6PM	2	18.75	15.4
15	2016-03	8767659	Thursday	7PM	2	19.19	15.71

In this result, we can see a clear decline in the amount of taxi use from 2015 to 2022 with monthly trip counts coming down from more than 8 million trips to just over 2 million trips. We can also see that the evening time is when most cabs are hailed, especially around 6PM. There is also a steady increase in the average total amount.

These can be seen with the below visualisations:

a. Most frequent day of the week

	most_trips_day_of_week	count
1	Friday	21
2	Thursday	21
3	Saturday	21

b. Most frequent time of day

	most_trips_hour_of_day	count
1	6PM	52
2	9PM	19
3	7PM	8

Recommendation: Increase number of cabs between 6PM to 9PM especially between Thursday to Saturday.

3.2 Business Question 2:

b. What was the average, median, minimum and maximum trip distance in **km** ?
c. What was the average, median, minimum and maximum speed in **km per hour** ?

=> In a Single table/dataframe/output

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) ?

The result of Question 2 consists of the average, medium, minimum, and maximum data of trip distance, speed, and trip duration. The minimum and maximum of these parameters are the filtered limits which help in verifying the effectiveness of our preprocessing. The results for the same are as follows:

	taxi_color	average_duration	median_duration	min_duration	max_duration	average_distance	median_distance	mi
1	green	17.63	14.4	2.4	149.4	6.33	4.51	
2	yellow	18.38	15	2.4	149.4	6.42	4.02	

3.3 Business Question 3

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:
    a. What was the total number of trips ?
    b. What was the average distance ?
    c. What was the average amount paid per trip (using *total_amount*) ?
    d. What was the total amount paid (using *total_amount*) ?

The answer comprises of the total number of trips, average distance, average amount paid per trip, and total amount paid for each hour, of each day, of each month of each pair of pickup and drop-off locations. The resulting dataframe is a large dataframe of 95480 rows. The result is given below:

	taxi_color	pickup_borough	dropoff_borough	month_name	day_of_week_name	time_of_day	total_trips	average
1	yellow	Queens	Manhattan	April	Friday	10AM	13783	
2	yellow	Queens	Manhattan	April	Friday	9AM	12247	
3	yellow	Manhattan	Manhattan	April	Friday	9AM	212466	
4	yellow	Queens	Queens	April	Friday	11AM	3379	
5	yellow	Manhattan	Brooklyn	April	Friday	4AM	5986	
6	yellow	Manhattan	Bronx	April	Friday	6AM	1060	
7	yellow	Queens	Queens	April	Friday	10AM	3671	
8	yellow	Brooklyn	Bronx	April	Friday	1AM	22	
9	yellow	Bronx	Brooklyn	April	Friday	8AM	20	
10	yellow	Manhattan	Manhattan	April	Friday	2AM	57309	
11	yellow	Manhattan	Bronx	April	Friday	1AM	2099	
12	yellow	Queens	Staten Island	April	Friday	6PM	33	
13	yellow	Queens	Queens	April	Friday	11PM	5263	
14	yellow	Manhattan	Manhattan	April	Friday	4PM	190747	
15	yellow	Queens	Bronx	April	Friday	12PM	275	

The 3 boroughs with the highest number of pickups were as follows:

	pickup_borough	total_trips_count
1	Manhattan	387927222
2	Queens	46609022
3	Brooklyn	23744421

The taxi companies can make sure large number of cabs are made available in these boroughs, Thursday to Saturday between 6PM to 9PM

3.4 Business Question 4

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

The overall percentage of trips where the drivers received tips are as follows:

Table

	1.2 percentage_tips
1	65.82

3.5 Business Question 5

5. Percentage trips where drivers received tips more than $5

The percentage of trips where the drivers received tips of more than $5 is as follows:

Table

	1.2 percent_tips
1	16.52

3.6 Business Question 6

6. Classify each trip into bins of durations:
    a. Under 5 Mins
    b. From 5 mins to 10 mins
    c. From 10 mins to 20 mins
    d. From 20 mins to 30 mins
    e. From 30 mins to 60 mins
    f. At least 60 mins

    Then for each bins, calculate:
    a. Average speed (km per hour)
    b. Average distance per dollar (km per $)

=> In a Single table/dataframe/output

The 6th business question consists of 6 different bins split in accordance with the duration of each trip. For each bin, the average speed and the average distance per dollar has been calculated. The results have been displayed below:

	trip_duration_bin	1.2 average_speed_kmph	1.2 average_distance_per_dollar
1	under 5 minutes	32.66	0.27
2	5 to 10 minutes	21.6	0.26
3	10 to 20 minutes	18.76	0.27
4	20 to 30 minutes	20.05	0.3
5	30 minutes to 1 hour	24.57	0.36
6	Over 1 hour	22.3	0.47

3.7 Business Question 7

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

Finally, the bin that would maximise the taxi driver's income would be **trips of duration 5 to 10 minutes.** This is because the average distance the driver needs to travel per dollar earned is the lowest in this bin.

4.0 Machine Learning

1. Build at least two different ML models (two different algorithms) using Spark ML pipelines + a baseline model to predict the **Total amount** of a trip:
    a. Build a baseline model by using the answer of Part 2 Q3c (average paid) and calculate its RMSE.
    b. Use all data except **October/NovemberDecember 2022** to train and validate your models and use the RMSE score to assess your models.
    c. Choose your best model and explain why you choose it (processing time, complexity, accuracy, etc).
    d. Using your best model, predict the **October/NovemberDecember 2022** trips and calculate the RMSE on your predictions. Does your model beat the baseline model

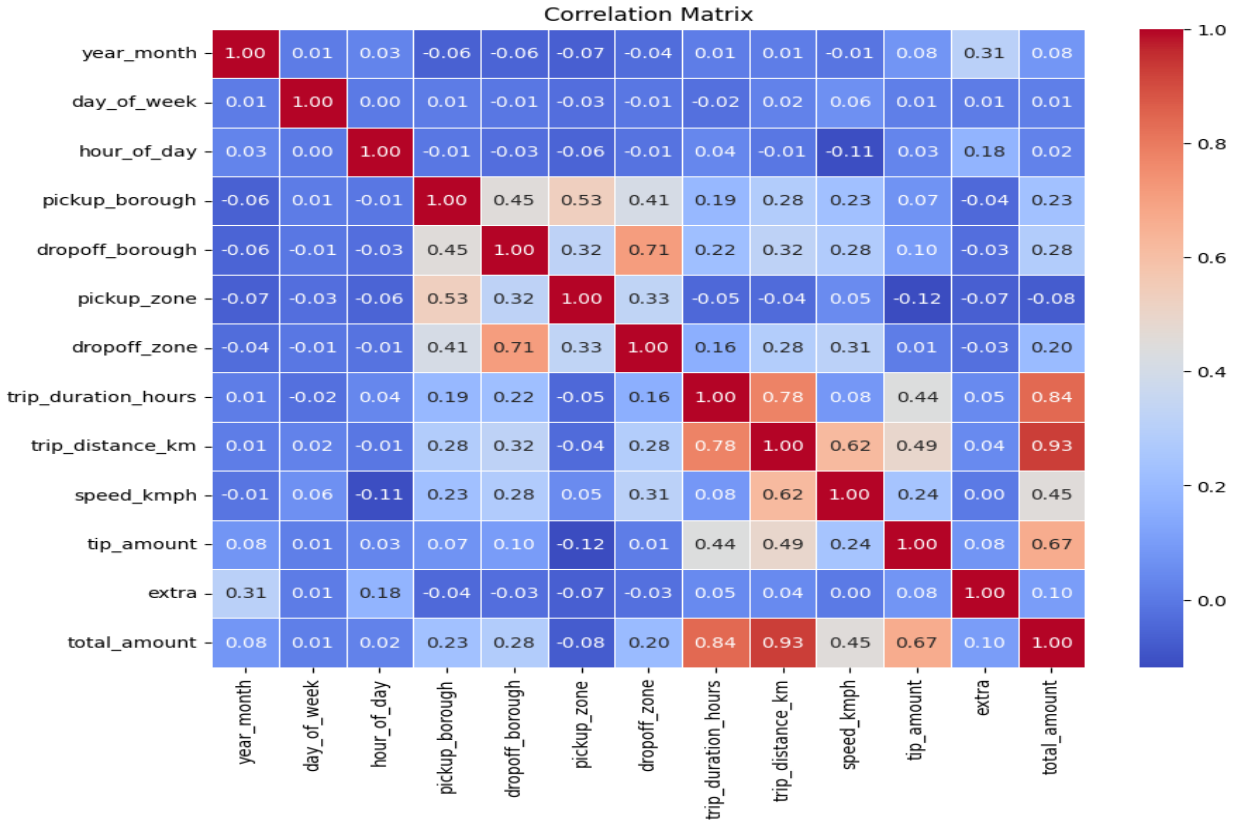
4.1 Machine Learning Data Preparation

The data preparation for Machine learning consists of the steps highlighted below.

1. Based on the assessment brief, the rows containing data from October to December 2022 are filtered out.
2. Following this filtration, the fare amount and the tolls amount column is dropped.
3. Due to memory and time constraints, the overall dataset is sampled by choosing 100 thousand rows at random to be the representative sample set.
4. The categorical columns are then converted to numeric features using String Indexing. These columns are as follows:

```
# Select features - Passenger count dropped, MTA & improvement dropped
features = ['year_month', 'day_of_week', 'hour_of_day', 'pickup_borough','dropoff_borough', 'pickup_zone', 'dropoff_zone',
            'trip_duration_hours', 'trip_distance_km', 'speed_kmph', 'tip_amount', 'extra', 'passenger_count', 'extra',
            'mta_and_improvement', 'total_amount']
```

Following column encoding, a correlation matrix is generated

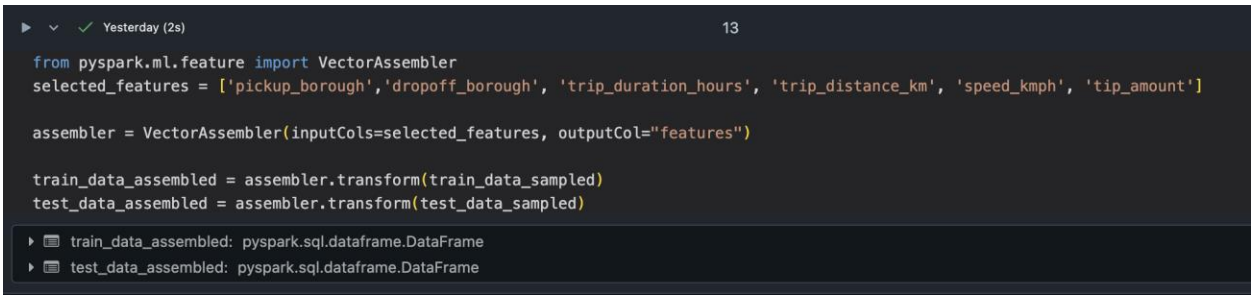


The correlation matrix shows that the trip duration, trip distance, speed, tip amount, pickup borough, and drop-off borough have the highest correlation with the total amount.

5. The data is split into the training and the testing datasets with the split being 70% if data into the training set and 30% into the testing set.

```
▶  ✓ 06:26 PM (<1s)                                                    7
train_data_sampled, test_data_sampled = taxi_ml_sampled.randomSplit([0.7, 0.3], seed=42)
▸ train_data_sampled:  pyspark.sql.dataframe.DataFrame = [lpep_pickup_datetime: timestamp, lpep_dropoff_datetime: timestamp ... 18 more fields]
▸ test_data_sampled:  pyspark.sql.dataframe.DataFrame = [lpep_pickup_datetime: timestamp, lpep_dropoff_datetime: timestamp ... 18 more fields]
```

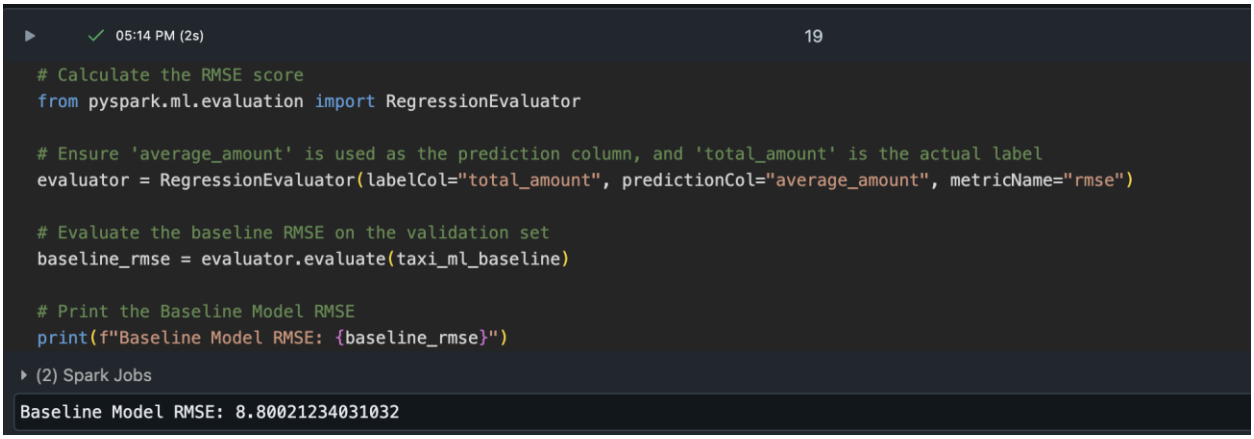
6. The chosen features are then converted into a vector using vector assembler. Vector Assembler is a transformer that combines a given list of columns into a single vector column. It is useful for combining raw features and features generated by different feature transformers into a single feature vector, to train ML models. [3]



7. Since the numeric columns represent different features, scaling is performed on these columns to bring them into a single scale.

4.2 Baseline Model

For the baseline model, as calculated in business question 3, the average trip amount for each zone is calculated and applied into an average amount column. The evaluation parameter for the entire project is chosen as **Root Mean Squared Error (RMSE)**. This average amount is used as the prediction feature and the RMSE score is calculated over for this average column against the total amount column.



As seen, the Baseline model RMSE equates to 8.8

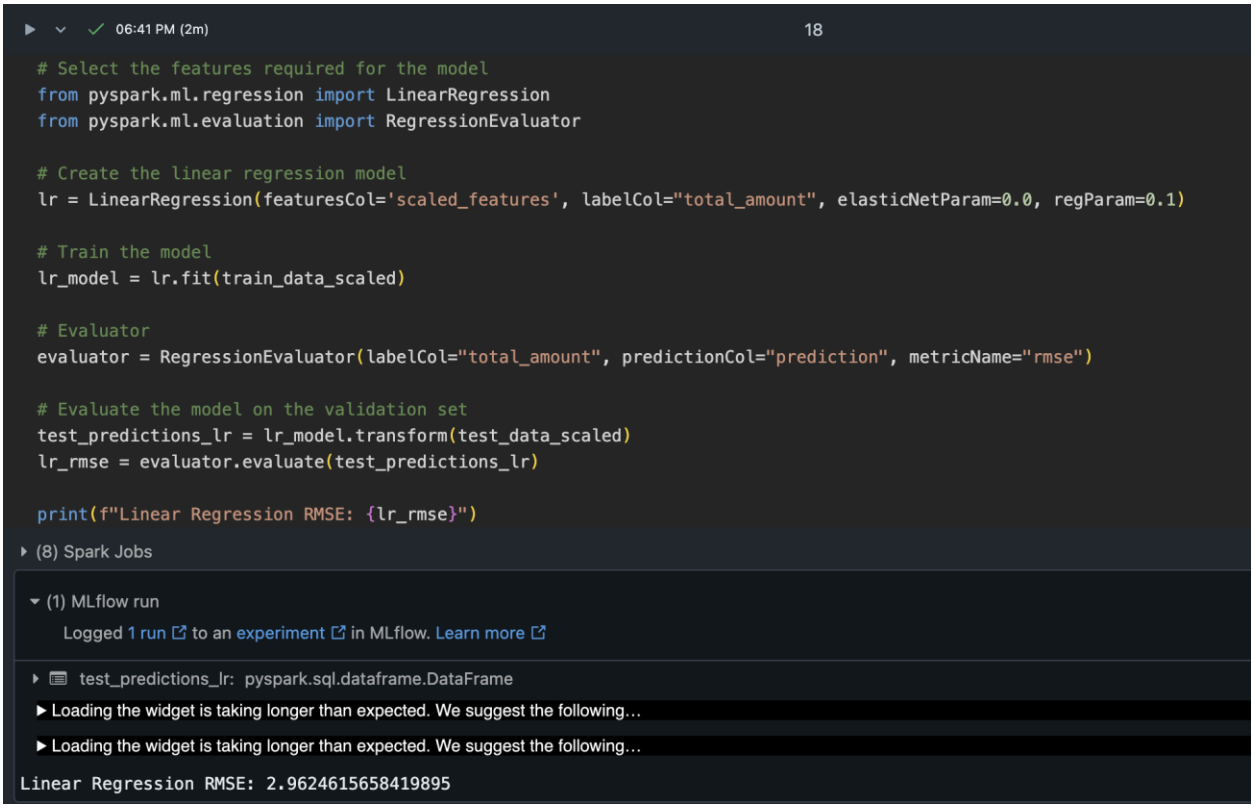
4.3 Modelling

4.3.1 Linear Regression

The first machine learning model chosen is Linear Regression. The reasoning for this is as follows:

- Linear Regression is simple to implement and easier to interpret the output coefficients.
- It is the best to use because of its low complexity compared to other algorithms when you know the relationship between the independent and dependent variable is a linear relationship
- Linear Regression is susceptible to over-fitting, but it can be avoided using some dimensionality reduction techniques, regularization (L1 and L2) techniques and cross-validation. [4]

We also make use of Ridge regularisation by setting elasticNetParam to 0.0 and set the strength of the regularisation to 0.1.



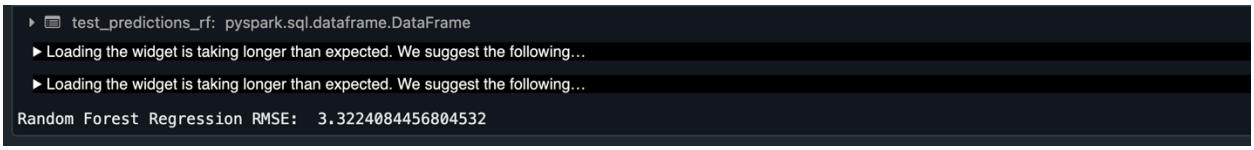
The RMSE of Linear regression comes down drastically from the baseline model and is 2.96.

4.3.2 Random Forest Regression

The second model implemented is the Random Forest regression model. The reasoning for selecting this model is as follows:

- It uses multiple Decision Trees to make predictions and can model complex, nonlinear relationships between features and target variables.
- It is subject to lower overfitting and higher accuracy due to existence of multiple decision trees.
- Is robust in data handling and can handle a wide array of numerical and categorical data including being good at dealing with outliers.

Random Forest result:



The RMSE of Random Forest at 3.32 is slightly higher than linear regression but still performs exceedingly well when compared to the baseline mode.

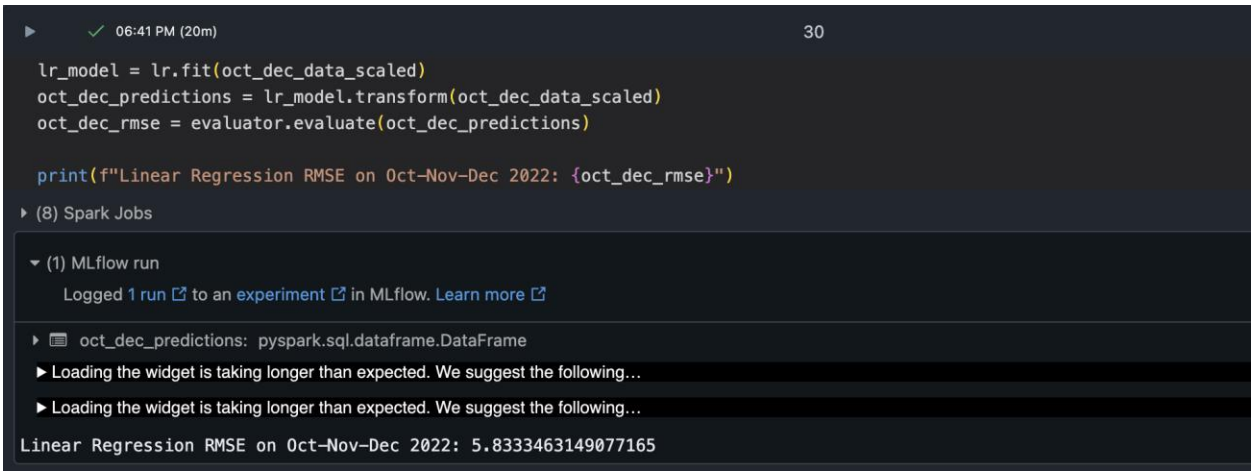
The best Model: Comparing the two RMSE scores, the better model of the two is Linear Regression. The reasoning for chosing the Linear Regression model were as follows:

- Better RMSE score
- Easy interpretability
- Faster convergence time

Conclusion on Modeling: It is seen that the linear regression model performs better than the Random Forest model with a lower RMSE score. The score is much lower than the baseline RMSE which indicates the model is performing better than the simple average.

4.4 Prediction of October, November, December trips

The October, November, and December data is used as unseen data for the model to perform predictions on. This data undergoes the same preprocessing as the rest of the data, i.e. dropping rows with null values, encoding, vector assembly, and scaling. Following the preprocessing steps, the linear regression model is applied onto this unseen data. The results are as follows:



The RMSE score is calculated to be 5.83 which is an increased score as compared to the general Linear Regression model but still performs better than the baseline model.

This model can be used as a baseline study for further implementations with higher compute resource availability where a larger amount of data can be utilised for modeling,

References:

[1] : New York/Speed Limits – Wazeopedia - https://www.waze.com//wiki/USA/New_York/Speed_limits#:~:text=The%20NYS%20statewide%20maximum%20speed,25%20mph%20unless%20otherwise%20posted.

[2] : https://www.nyc.gov/html/dot/downloads/pdf/current-pre-vision-zero-speed-limit-maps.pdf

[3] : Vector Assembler – Data Science with Apache Spark -

https://george-jen.gitbook.io/data-science-and-apache-spark/vectorassembler

[4] : ML – Advantages and Disadvantages of Linear Regression – 30 Mar 2023 - https://www.geeksforgeeks.org/ml-advantages-and-disadvantages-of-linear-regression/